2021

ITERATIVE METHODS FOR COMPUTING A FEW EIGENPAIRS OR SINGULAR TRIPLETS OF LARGE SPARSE MATRICES

Jennifer R. Picucci
University of Rhode Island, jenniferpicucci@uri.edu

ITERATIVE METHODS FOR COMPUTING A FEW EIGENPAIRS OR SINGULAR TRIPLETS OF LARGE SPARSE MATRICES

BY

JENNIFER R. PICUCCI

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

APPLIED MATHEMATICS

UNIVERSITY OF RHODE ISLAND

2021

DOCTOR OF PHILOSOPHY DISSERTATION

OF

JENNIFER R. PICUCCI

APPROVED:

Dissertation Committee:

Major Professor James Baglama

Vasilije Perović

Richard Vaccaro

Brenton DeBouf
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2021

ABSTRACT

This thesis presents new hybrid restarted Lanczos methods for computing eigenpairs and singular triplets of large matrices. Our methods combine thick-restarting with Ritz or harmonic Ritz vectors with iteratively refined Ritz vectors to compute a few of the extreme eigenpairs of symmetric matrices or singular triplets of rectangular matrices. The refined process improves the (harmonic) Ritz values/vectors yielding better approximations, i.e., this process results in a "smaller" residual norm compared to just using Ritz/harmonic vectors. The iterative refined process we developed improves the refined values/vectors by using a scheme, where we replace the approximate eigenvalue/singular value in the original refined scheme with the latest computed refined Ritz value until convergence. The thick-restarting schemes are superior in reference to efficiency to other restarted schemes, but are not available when using refined or iterative refined Ritz vectors. Therefore, we developed hybrid restarted methods that switch between the efficient thick-restarted scheme and restarting with a linear combination of "the better approximating" iterative refined Ritz vectors. Our developed methods have shown to be very effective on small subspaces, i.e., when memory is limited. We provide many theoretical results and numerical examples.

ACKNOWLEDGMENTS

I would first like to thank my amazing thesis advisor, Dr. James Baglama. I cannot express enough gratitude for Dr. Baglama's patience and encouragement throughout the past few years in order to complete this research while I was working full time. I know I have frequently been pulled in more directions than anticipated which delayed our research more than planned.

For their time and commitment to my success, I would also like to thank my doctoral committee. I am grateful to Dr. Vasilije Perović for not only serving on my committee, but also for being my graduate Linear Algebra instructor here at URI. Dr. Perović was an enjoyable lecturer on the subject and passed his enthusiasm for the subject to anyone in attendance. I am also very appreciative of Dr. Richard Vaccaro from the Electrical Engineering Department for accepting my request to serve on my committee and Dr. Christopher Baxter, from the Ocean Engineering Department, for agreeing to be the chair of my doctoral defense. Lastly, I would like to thank Dr. Tom Bella for assisting in the writing and development of the first journal article in this thesis.

Returning back to school after working in government research for 10 years was harder than I ever imagined. I would not have succeeded without the friends I discovered in the Mathematics graduate Department while here at URI. These individuals helped me through every stage of this process, especially Eric Peterson and Sarah Van Beaver in joining forces to study for our comprehensive exams. They provided not only sounding boards for discussion but many many hours of much needed laughs as well. It goes without saying that this thesis was an enormous adventure to take on while working; as such, this is an achievement I am very proud of accomplishing. This effort would not have been possible without everyone in my life who has helped me along the way, my Fellowship of the Ring.

I am very thankful for the opportunity to complete Long Term Training through

my work at the U.S. Army Corps of Engineers, Engineer Research and Development Center. This allowed me to take a full year away from work responsibilities to complete my course work and oral examination. Without the ability to focus solely on academic responsibilities I don't believe I could have accomplished everything necessary to finish my research and graduate. My friends Mihan and Oliver Taylor, Leigh and Neil Williams, and Pam Kinnebrew from ERDC were a constant support holding me up when the effort threatened to knock me down.

Last, but never least, I need to thank my family. They endured constant explanations and discussions into mathematics that gave them headaches. They patiently supported me when I stayed up all night to write and doodled matrices and equations on napkins at parties. I know this entire process seemed crazy to them but they helped me along solely because it was important to me. I cannot begin to tell them how much that means to me.

PREFACE

This thesis will be presented in manuscript format. In Chapter 1 we provide the motivation and goals for this research along with a synopsis of the following chapters. Chapter 2 is the first manuscript, published on May 4, 2021 to the SIAM Journal on Scientific Computing. Chapter 3 is the second manuscript submitted July 30, 2021 to Numerical Algorithms. Chapter 4 provides the conclusions for this research and Appendix A contains the MATLAB codes used for numerical examples.

Dr. James Baglama was involved in the idea, design, and writing, of both research papers and codes. Dr. Tom Bella and Dr. Vasilije Perović assisted in providing ideas, comments and writing of the first and second research papers, respectively.

A list of references used for the respective manuscripts is provided at the end of each chapter.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The motivation for this thesis pertains specifically to my job at the U.S Army Corps of Engineers, Engineer Research and Development Center (ERDC) where I am a research Mathematician. At ERDC, I have spent over 10 years working with a seismic sensor system and its data processing algorithms. These sensors use a classification method which looks for characteristics that eliminate each option and deem whatever is left as the activity detected. This can cause misclassifications for unknown items not in the system database. If those misclassifications are classified as a threat activity this creates the potential for unnecessary resources to be expended in order to investigate the event. For this reason, we were looking for a new way to add confidence bounds to each of the items in the classification database. This would allow for items too far outside that bound to be given the label unknown instead of whichever classification option was left at the end of the process.

The method we chose to explore the use of confidence bounds was that of Principal Component Analysis (PCA) by way of Partial Singular Value Decomposition (PSVD). PCA has been used in multiple fields to filter outliers of a population based on specific data features. For the seismic sensor, we sometimes have a delay in classification of vehicles if they are on a heavily rutted road as the potholes can seem like digging. With PCA these signals from vehicles even with potholes in the road should fall outside a chosen boundary line and can be correctly classified more quickly. But in order to implement PCA we needed to develop a fast PSVD method. As these system may not have sufficient storage capabilities, in addition to increased speed this new method is required to run using limited data and therefore, the focus of this thesis is on the development of a PSVD method.

While studying and developing a new PSVD method we came across an ability to add an iterative cycle to improve the refined scheme developed by Jia [2] for a faster method. We quickly realized that a more concrete theoretical basis was needed to justify our new algorithms. While developing these foundations we explored the connection of the SVD of a matrix and the equivalent symmetric eigenvalue problems. More specifically, if $A \in \mathbb{R}^{m \times n}$, then $A^T A$ is a symmetric matrix which we refer to as the normal system. The eigenvalues, σ_j^2 of $A^T A$ are the squares of singular values of A, while the associated eigenvectors, v_j of $A^T A$ are the corresponding right singular vectors of A. The left singular vectors, u_j are the computed as $u_j = \frac{1}{\sigma_j} A v_j$, $(\sigma_j \neq 0)$. A similar link can be found with the symmetric matrix $C = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$, which we refer to as the augmented system. For C, the eigenvectors corresponding to eigenvalues $\pm\sigma_j$ are $\frac{1}{\sqrt{2}}[u_j^T \, , \pm v_j^T]^T$, where $\{\sigma_j, u_j, v_j\}$ is a singular triplet of A. These connections allowed us to use the simpler framework of the symmetric case to prove our iterative process converges and why it provides a "smaller" residual, i.e., better approximation, than that of the refined method but would not work very well when used as starting vectors for restarting methods. This led us to the development of the hybrid portion of our new method and its implementation.

Chapter 2 contains the journal article, *Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix*. In this paper, we first provide background on thick-restarted with Ritz vectors and the refined Ritz methods along with a new way to represent the refined residual vectors under the Lanczos relationship. We then present our new hybrid algorithm along with convergence results and justifications for the use of linear combinations of iterative refined Ritz vectors. Lastly in this paper, we provide numerical results to compare the new method to multiple other methods currently available.

In Chapter 3, we make the connection from symmetric eigenvalue problems to

PSVD and introduce the second journal article *Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods*. It extends the hybrid method from the first paper to rectangular matrices by utilizing the Golub Kahan Lanczos Bidiagionalization (GKLB) method for explicitly computing the PSVD. This journal article uses the theoretical foundation provided in the first article along with relationships of the normal and augmented systems to build the needed methods to develop algorithms that are useful in the future for my work applications. We conclude with numerical results. Chapter 4 follows with a Conclusion section summing up the results from the entire thesis. Finally, Appendix A includes the MATLAB codes used in the numerical examples.

List of References

[1] J. Demmel, *Applied Numerical Linear Algebra*, Theorem 3.3 SIAM, (1997), pp. 110-133, 195.

[2] Z. Jia, *Refined iterative algorithms based on arnoldi's process for large unsymmetric eigenproblems*, Linear algebra and its applications, 259 (1997), pp. 1–23.

[3] D. C. Sorensen, *Implicit application of polynomial filters in a k-step arnoldi method*, Siam journal on matrix analysis and applications, 13 (1992), pp. 357–385.

CHAPTER 2

Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix

James Baglama, Tom Bella, and Jennifer Picucci

Department of Mathematics

University of Rhode Island, Kingston, Rhode Island 02881-0816, USA

This manuscript was published to the SIAM Journal on Scientific Computing on May 4, 2021 and can be found at https://doi.org/10.1137/20M1344834.

Keywords:

restarted Lanczos method, eigenvalue computation, singular value, refined Ritz, thick–restarted.

AMS

65F15, 65F50, 15A18

Abstract We developed a hybrid restarted Lanczos method that combines thick–restarting with Ritz vectors with iteratively refined Ritz vectors to compute a few of the extreme eigenvalues and associated eigenvectors of a large sparse symmetric matrix A. The iterative refined Ritz vectors use a scheme, where we replace the approximate eigenvalue in the original refined scheme with the latest computed refined Ritz value until convergence. The thick–restarting schemes have shown to be superior to most other schemes, particularly restarted schemes of linear combinations. However, the simple thick–restarting Lanczos scheme is not available when using refined or iterative refined Ritz vectors. Instead, we use a hybrid restarted scheme that switches between thick–restarted with Ritz vectors and restarting with a judiciously chosen linear combination of iterative refined Ritz vectors. We provide some theoretical results and several computed examples.

2.1 Introduction

Large sparse symmetric eigenvalue problems

$$Ax = \lambda x \qquad A \in \mathbb{R}^{n \times n} \tag{1}$$

are some of the most important and profoundly studied areas in numerical linear algebra. Although these problems are numerically attractive, they can exhibit computational challenges for even the best modern routines, e.g. clustering of eigenvalues, matrix sizes (memory constraints), and orthogonality. The importance of these problems and computational challenges have spurred a considerable amount of research, e.g.[1, 2, 6, 7, 14, 22, 23, 27, 29, 30] and references within. The goal of this paper is twofold: the development of a new, simple algorithm that uses as little storage as possible to compute a few of the extreme eigenpairs of a large sparse symmetric matrix A, and provide some insightful results on the (iterative) refined Ritz scheme. Since A is so large, we assume its factorization is not feasible and only the evaluation of matrix–vector products with the matrix A is available.

The motivation for the paper starts with the pioneering algorithm of Sorensen [22] for symmetric matrices called the Implicitly Restarted Lanczos (IRL) method (non–symmetric case is referred to as Implicitly Restarted Arnoldi (IRA) method). The IRL method is a restarted Krylov subspace method that implicitly modifies the starting vector p_1 with an accelerating polynomial. The accelerating polynomial is determined by a specific selection of zeros of the polynomial, also called shifts. There are several choices for shifts; the IRL method in [22] uses Ritz values as shifts. The choice of shifts is crucial for performance of the IRL method and there have been investigations into other choices, e.g. Leja points [1] and harmonic Ritz [16]. In 1997, Jia [6] introduced the concept of refined Ritz vectors. The idea is for a given approximate eigenvalue μ of A, to minimize $\|Az - \mu z\|$ for a unit vector z from the current working Krylov subspace. Refined Ritz vectors often provide better eigenvector ap-

proximations than the Ritz vectors, see analysis [9, 11] for details. Since refined Ritz will be the basis of our new method, a thorough discussion is presented in Section 2.4. In [7], Jia applied the refined concept in combination with the IRA procedure, referred to as Implicitly Restarted Refined Arnoldi method (IRRA). Morgan [15] showed the equivalence of the IRA method with Ritz values as shifts with augmenting the Krylov subspace by certain Ritz vectors. Wu and Simon [27] exploited this idea in the symmetric case, and by using the property that all of the Ritz vectors are multiples of same residual vector, they created a simple augmented method, called the "Thick–Restarted Lanczos method." The method is mathematically equivalent to the IRL method and avoids the need for the implicitly shifted QR algorithm. Similarly, Stewart [26] presents the Krylov–Schur method for the non–symmetric case and showed its equivalence to, and numerical superiority over, the IRA method.

The key feature needed with the thick–restarting Lanczos method with Ritz vectors is that the resulting space remains a Krylov subspace, which is possible since the Ritz vectors are all multiples of a single residual vector, [15]. The simple thick–restarting scheme by Wu and Simon [27] with refined Ritz vectors in place of Ritz vectors is not possible, because refined Ritz vectors are not all multiples of a single residual vector, see Theorem 2.4.3 in Section 2.4. Furthermore, as discussed in [9, 11, 18] on the relationship between refined Ritz and Ritz vectors they are not parallel unless refined Ritz vectors equal eigenvectors. We present a similar result for the context here, see Theorems 2.4.2 and 2.5.2.

To improve approximations to desired eigenpairs and decrease the refined Ritz residuals norm, we introduced an iterative scheme, where we replace the approximate eigenvalue in the refined scheme with the latest computed refined Ritz value until convergence, Section 2.5. This process has the added benefit of eliminating part of the refined Ritz residuals. The resulting residual vector has convenient qualities and a

"smaller" norm. However, like refined Ritz vectors, the iterative refined Ritz vectors are not all multiples of a single residual vector, see Section 2.5.

Morgan showed in [15] that the IRA method developed by Sorensen can be implemented by using a starting vector with a cleverly chosen linear combination of the desired Ritz vectors. We use a similar linear combination when restarting with the iterative refined Ritz vectors. The idea is to inherit similar, beneficial, restarting properties.

However, when using only refined Ritz or iterative refined Ritz vectors for restarting we observed examples of stagnation, erratic convergence, or very slow convergence. Slow convergence is exacerbated with restarted methods when using a low dimensional subspace and/or clustered eigenvalues, see Examples 5.2 and 5.3 in Section 2.5. Thick–restarting with Ritz vectors also exhibits very slow convergence when using a low dimensional subspace. Therefore, we implemented a hybrid method Section 2.6 that depends on certain criteria for switching from thick–restarting with Ritz vectors to restarting with a linear combination of iterative refined Ritz vectors. We observed through numerical experiments that although switching from thick–restarted Ritz to a linear combination of iterative refined results in a temporary undesirable spike in the norms of the residuals, this often relieves the stagnation/slow convergence, resulting in an overall faster convergence. This has the added benefit of being able to use a small Krylov subspace, see Figure 2.

Hybrid methods have been used previously for combining other forms of eigenvector approximations. For example, a comparable method is a block hybrid method that was proposed in [12] in which thick–restarting is performed by "modified" Ritz vectors, computed over a block Krylov subspace. Their block hybrid algorithm uses a power method with refined Ritz vectors "stitched" together with thick–restarting with modified Ritz vectors. Although absent with their code, we do provide an example

in Section 2.7 on the symmetric matrix experiment presented in their paper.

It should be noted that the Jacobi–Davidson method and extensions thereof are competitive for the computation of a few eigenvalues and associated eigenvectors, particularly when a known preconditioner is available; see [5, 21, 24, 23, 29] for descriptions of such methods. We do provide some comparisons in Section 2.7.

Throughout this paper $\|\cdot\|$ denotes the Euclidean vector norm or the associated induced matrix norm. When useful and for ease of presentation we will utilize MATLAB notation.

The subsequent parts of the paper are organized as follows. We begin with a background of restarted Lanczos method Section 2.2, thick–restarting with Ritz vectors in Section 2.3, and refined Ritz pairs in Section 2.4. Some theoretical results and relationships on refined Ritz pairs are included in Section 2.4. In Section 2.5 we describe our new strategy for iterative refined Ritz vectors and provide several theoretical results, motivation, and relationships. Our new hybrid method for computing the extreme eigenpairs is presented in Section 2.6. Numerical examples are presented in Section 2.7 and conclusions follow in Section 2.8.

2.2 Lanczos Method

For this discussion, we describe a Lanczos method that is modeled after the algorithms presented in [17, 27]. Given a unit vector p_1, we define the Krylov subspace

$$\mathbb{K}_m(A, p_1) = \text{span}\{p_1, Ap_1, A^2p_1, \ldots, A^{m-1}p_1\}. \tag{2}$$

Application of m steps of the MLan(0) Algorithm 2.1 given below with the initial starting vector p_1 applied to the symmetric matrix A generates a sequence of m orthogonal vectors $p_j \in \mathbb{R}^n$ that form an orthonormal basis for the Krylov subspace (2). This algorithmic process forms the well–known Lanczos decomposition,

$$AP_m = P_mT_m + fe_m^T, \tag{3}$$

where the Lanczos vectors

$$P_m \;=\; [p_1, p_2, \ldots, p_m] \in \mathbb{R}^{n\times m} \tag{4}$$

satisfy $P_m^T P_m = I_m$, and $f \in \mathbb{R}^n$, referred to as the residual vector, satisfies $P_m^T f = 0$. The matrix I_m denotes the identity matrix of order m and the vector $e_m \in \mathbb{R}^m$ consists of the last column of I_m. The matrix T_m is tridiagonal of the form

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & & \ddots & \\ & & \ddots & \ddots & \beta_{m-1} \\ 0 & & & \beta_{m-1} & \alpha_m \end{bmatrix} \in \mathbb{R}^{m\times m}. \tag{5}$$

For brevity we give a more general version of MLan(ℓ) that will be used in Section 2.3.

Algorithm 2.1 MLan(ℓ)

1: **Input:** $A \in \mathbb{R}^{n\times n}$: symmetric matrix,
$p_1, \ldots, p_{\ell+1} \in \mathbb{R}^{n\times(\ell+1)}$: orthonormal vector(s) (10),
$\bar{T}_{\ell+1} \in \mathbb{R}^{\ell+1\times\ell+1}$: if $\ell > 0$ input matrix (13),
m : maximum number of Lanczos vectors.
2: **Output:** $P_m = [p_1, p_2, \ldots, p_m] \in \mathbb{R}^{n\times m}$: orthogonal matrix (4) or (15),
$T_m \in \mathbb{R}^{m\times m}$: if $\ell = 0$ tridiagonal matrix (5), if $\ell > 0$ matrix (16),
$f \in \mathbb{R}^n$: residual vector.
3: **for** $j = \ell + 1 : m$ **do**
4: $f := Ap_j$
5: **if** $j > \ell + 1$ **then** $f := f - p_{j-1}\beta_j$
6: **if** $j = \ell + 1$ and $\ell > 0$ **then** $f := f - P_\ell \bar{T}(\ell+1, 1:\ell)^T$
7: $\alpha_j := f^T p_j$ and $f := f - p_j\alpha_j$
8: Reorthogonalization: $f := f - P_j(P_j^T f)$
9: **if** $j < m$ **then** $\beta_j := \|f\|$ and $p_{j+1} := f/\beta_j$

For our discussion, we will assume that we can perform $m - \ell$ steps of Algorithm 2.1 to generate the Lanczos tridiagonal decomposition (3) or, later, (14). In practice, near-breakdowns can occur, i.e. $\beta_j \approx 0$ for some j (step 9 Algorithm 2.1), see strategies such as those described in [2, 13] to continue the Lanczos process. Since

m in our discussion is of a modest size and the desired number of eigenpairs k is small, a near-breakdown without convergence is rare and therefore, we will assume for the following discussion, T_m is unreduced. The reorthogonalization step introduced in step 8 of Algorithm 2.1 is a simple strategy to help maintain orthonormality among independent vectors for modest values $m \ll n$. More robust reorthogonalization strategies, e.g. selective or partial are described in the literature, see e.g. [17, 20, 27].

Let $\{\theta_j, y_j\}_{j=1}^m$ be the m eigenpairs of the tridiagonal matrix T_m with the desired k eigenpairs appearing as the leading entries, i.e. $\{\theta_j, y_j\}_{j=1}^k$. Define $x_j := P_m y_j$. Then θ_j and x_j are commonly referred to as a Ritz value and a Ritz vector of A, respectively, or simply a Ritz pair. It follows from (3) that

$$Ax_j - \theta_j x_j \;\; = \;\; AP_m y_j - \theta_j P_m y_j = (P_m T_m - \theta_j P_m)\, y_j + f e_m^T y_j = f e_m^T y_j, \quad (6)$$

and therefore the norms of the residual errors for the Ritz pairs $\{\theta_j, x_j\}$ satisfy

$$\|Ax_j - \theta_j x_j\| \;\; = \beta_m |e_m^T y_j| \quad (7)$$

where $\beta_m = \|f\|$. For numerical experiments, we use a stopping criterion

$$\beta_m |e_m^T y_j| \;\; \leq \epsilon \|A\| \quad (8)$$

where ϵ is a user specified tolerance and $\|A\|$ is approximated by the eigenvalue of largest magnitude over all iterations thus far of the computed matrices T_m. For a given number k of desired eigenpairs we have convergence when the norm (7) for all $j = 1, \ldots, k$ is less than $\epsilon \|A\|$.

2.3 Thick–Restarted Lanczos with Ritz vectors

We next briefly describe the method of thick–restarting with Ritz vectors as outlined in [27] which is needed in subsequent sections. For a thorough description, we refer the reader to [25, 26, 27, 29] and the references within.

From (6), we have for $j = 1, \ldots, k$

$$Ax_j \;=\; \theta_j x_j + p_{m+1}\bar{\beta}_j \tag{9}$$

where $p_{m+1} = f/\beta_m$ and $\bar{\beta}_j = \beta_m e_m^T y_j$. Define

$$\bar{P}_k \;:=\; [x_1, \ldots, x_k] \quad \text{and} \quad \bar{P}_{k+1} := [\bar{P}_k, p_{m+1}]. \tag{10}$$

Then we have

$$A\bar{P}_k \;=\; \bar{P}_{k+1}\bar{T}_{k+1,k} \tag{11}$$

where

$$\bar{T}_{k+1,k} = \begin{bmatrix} \theta_1 & & & 0 \\ & \theta_2 & & \\ & & \ddots & \\ 0 & & & \theta_k \\ \bar{\beta}_1 & \bar{\beta}_2 & \ldots & \bar{\beta}_k \end{bmatrix}. \tag{12}$$

The factorization (11) can easily be extended to m vectors via MLan(k) Algorithm 2.1 by noticing that

$$\bar{P}_{k+1}^T A \bar{P}_{k+1} = \bar{T}_{k+1} = \begin{bmatrix} \theta_1 & & & 0 & \bar{\beta}_1 \\ & \theta_2 & & & \bar{\beta}_2 \\ & & \ddots & & \vdots \\ 0 & & & \theta_k & \bar{\beta}_k \\ \bar{\beta}_1 & \bar{\beta}_2 & \ldots & \bar{\beta}_k & \alpha_{k+1} \end{bmatrix} \tag{13}$$

where the last diagonal element α_{k+1} is computed in step 7 in the MLan(k) Algorithm 2.1. Step 6 in MLan(k) Algorithm 2.1 is used to ensure the orthogonalization of the newly computed Lanczos vector against $\bar{P}_k$ (10), c.f. Algorithm 3 in [27]. After $m-k$ steps of MLan(k) Algorithm 2.1 we have,

$$A\bar{P}_m \;=\; \bar{P}_m \bar{T}_m + \bar{f} e_m^T, \tag{14}$$

where

$$\bar{P}_m \;=\; [\bar{P}_{k+1}, p_{k+2}, \ldots, p_m] \in \mathbb{R}^{n \times m}, \tag{15}$$

satisfies $\bar{P}_m^T \bar{P}_m = I_m$, and $\bar{f} \in \mathbb{R}^n$ satisfies $\bar{P}_m^T \bar{f} = 0$ and

$$\bar{T}_m = \begin{bmatrix} \bar{T}_{k+1} & \beta_{k+1} & & & 0 \\ \beta_{k+1} & \alpha_{k+2} & \beta_{k+2} & & \\ & & & \ddots & \\ & & \ddots & \ddots & \beta_{m-1} \\ 0 & & & \beta_{m-1} & \alpha_m \end{bmatrix} \in \mathbb{R}^{m \times m}. \tag{16}$$

Algorithm 3.1 Thick–Restarted Lanczos

1: **Input:** $A \in \mathbb{R}^{n \times n}$: symmetric matrix,
$p_1 \in \mathbb{R}^n$: orthonormal vector,
k : number of desired eigenpairs of A,
m : maximum number of Lanczos vectors.
2: **Output:** (14) or $\{\theta_j, x_j\}_{j=1}^k$ approximate eigenpairs of A.
3: Compute factorization (3) by MLan(0) Algorithm 2.1
4: Compute the k desired eigenpairs $\{\theta_j, y_j\}_{j=1}^k$ of T_m (5) or $\bar{T}_m$ (16)
5: Check convergence (7)
6: Set up (11) and (13) and call MLan(k) Algorithm 2.1 to get factorization (14)
7: Goto 4

2.4 Refined Ritz vectors

We briefly describe and provide some results on refined Ritz vectors and values. For a thorough description, we refer the reader to [6, 7, 9, 11]. The refined Ritz vector z_j for an approximate desired eigenvalue μ_j of A is computed by

$$\min_{\substack{z_j \in \mathbb{K}_m(A,p_1) \\ \|z_j\|=1}} \|Az_j - \mu_j z_j\|. \tag{17}$$

Since $z_j \in \mathbb{K}_m(A, p_1)$, we can represent it in terms of the Lanczos vectors (4), i.e. $z_j = P_m w_j$ for some unit vector w_j. Given $\beta_m = \|f\|$, define

$$P_{m+1} := [P_m, p_{m+1}] \in \mathbb{R}^{n \times (m+1)}, \qquad T_{m+1,m} := \begin{bmatrix} T_m \\ \beta_m e_m^T \end{bmatrix} \in \mathbb{R}^{(m+1) \times m}, \tag{18}$$

where $p_{m+1} := f/\beta_m$. Let $I_{m+1,m}$ denote the first m columns of the identity matrix of order $m+1$. For each μ_j compute the smallest singular value $\sigma_{\downarrow j}$ and associated

unit singular vectors of $(T_{m+1,m} - \mu_j I_{m+1,m})$ i.e.

$$(T_{m+1,m} - \mu_j I_{m+1,m})v_j = \sigma_{\downarrow j} u_j \tag{19}$$

$$(T_{m+1,m} - \mu_j I_{m+1,m})^T u_j = \sigma_{\downarrow j} v_j. \tag{20}$$

We refer to the following unit vectors $v_j \in \mathbb{R}^m$ as the right singular vector and $u_j \in \mathbb{R}^{m+1}$ as the left singular vector associated with the smallest singular value $\sigma_{\downarrow j}$. Therefore,

$$\begin{aligned} \min_{\substack{z_j \in \mathbb{K}_j(A,p_1) \\ \|z_j\|=1}} \|Az_j - \mu_j z_j\| &= \min_{\|w_j\|=1} \|AP_m w_j - \mu_j P_m w_j\| \\ &= \min_{\|w_j\|=1} \|(P_{m+1}(T_{m+1,m} - \mu_j I_{m+1,m})w_j\| \\ &= \|(T_{m+1,m} - \mu_j I_{m+1,m})v_j\| = \sigma_{\downarrow j}. \end{aligned} \tag{21}$$

Then the refined Ritz vector z_j for μ_j is defined as $z_j := P_m v_j$. The approximate eigenvalue can be selected as the Ritz value θ_j or as the Rayleigh quotient $\rho_j = z_j^T A z_j = v_j^T T_m v_j$. Jia [7] suggested using ρ_j in place of θ_j as it may be more accurate. Setting $\mu_j = \theta_j$ and using (7), we have from [7, 9] that

$$\|Az_j - \rho_j z_j\| \leq \|Az_j - \theta_j z_j\| \leq \|Ax_j - \theta_j x_j\| = \beta_m |e_m^T y_j|. \tag{22}$$

If $\|Ax_j - \theta_j x_j\| \neq 0$, then $\|Az_j - \theta_j z_j\| < \|Ax_j - \theta_j x_j\|$. The following discussion establishes the left inequality for μ_j not necessarily $\mu_j = \theta_j$ and shows when a strict inequality exists. This sets the foundation for the subsequent section on iterative refined Ritz.

Using $\{\rho_j, z_j\}$ as the approximation, we have the following. Equate the first m rows of (19) and the last row to obtain

$$(T_m - \mu_j I_m)v_j = \sigma_{\downarrow j} u_{j(1:m)} \tag{23}$$

$$\beta_m e_m^T v_j = \sigma_{\downarrow j} u_{j(m+1)} \tag{24}$$

and left multiply (23) by v_j^T to obtain

$$\mu_j - \rho_j \;=\; -\sigma_{\downarrow j} v_j^T u_{j(1:m)}. \tag{25}$$

Then using the relationships (23)–(25) we have,

$$\begin{aligned} Az_j = AP_m v_j &= P_m T_m v_j + f e_m^T v_j \\ &= \rho_j z_j + \sigma_{\downarrow j} P_{m+1}(u_j - ([v_j;0]^T u_j)[v_j;0]) \end{aligned} \tag{26}$$

where $[v_j;0] \in \mathbb{R}^{m+1}$. Using (26) we have,

$$\begin{aligned} \|Az_j - \rho_j z_j\| &= \sigma_{\downarrow j}\|P_{m+1}u_j - ([v_j;0]^T u_j)P_{m+1}[v_j;0]\| \\ &= \sigma_{\downarrow j}\|u_j - ([v_j;0]^T u_j)[v_j;0]\| \\ &\leq \sigma_{\downarrow j}. \end{aligned} \tag{27}$$

The inequality in (27) comes from using the orthogonal projection, as summarized in Lemma 2.8.1 in the appendix. A strict inequality $\|Az_j - \rho_j z_j\| < \sigma_{\downarrow j}$ exists if $\mu_j \neq \rho_j$ then $[v_j;0]^T u_j \neq 0$ in (25), also see Lemma 2.8.1.

Notice from (25) and (26) we have

$$\begin{aligned} Az_j = AP_m v_j &= (\rho_j - \sigma_{\downarrow j} v_j^T u_{j(1:m)}) z_j + \sigma_{\downarrow j} P_{m+1} u_j \\ &= \mu_j z_j + \sigma_{\downarrow j} P_{m+1} u_j \end{aligned} \tag{28}$$

and $\|Az_j - \mu_j z_j\| = \sigma_{\downarrow j}\|P_{m+1}u_j\| = \sigma_{\downarrow j}$. Therefore, we have

$$\|Az_j - \rho_j z_j\| \;\leq\; \|Az_j - \mu_j z_j\| \tag{29}$$

and if $\mu_j \neq \rho_j$ then $\|Az_j - \rho_j z_j\| < \|Az_j - \mu_j z_j\|$. The relationship and convergence properties of the refined Ritz vector z_j and Ritz vector x_j are described in detail in [9, 11, 18]. In particular, they show, when $\sigma_{\downarrow j} = 0$, the vectors z_j and x_j are parallel to an eigenvector of A and $\rho_j = \theta_j$ is an exact eigenvalue of A. It is remarked in [18] that for a special, non–symmetric case, that z_j and x_j can be parallel, even though

$\sigma_{\downarrow j} \neq 0$. However, this is not the case in the context here for the symmetric problem, as Theorem 2.4.2 below demonstrates.

Corollary 2.4.1. *Let T_m (5) be unreduced and $\beta_m \neq 0$. Given the singular value relationships (19) and (20) we have*

i.) $\sigma_{\downarrow j} \neq 0$ and

ii.) $u_j \neq \pm e_{m+1}$.

Proof. i.) To show that $\sigma_{\downarrow j} \neq 0$, we argue via contradiction. Let $\sigma_{\downarrow j} = 0$, then from (23) and (24) we have,

$$(T_m - \mu_j I_m)v_j = 0 \tag{30}$$

$$\beta_m e_m^T v_j = 0. \tag{31}$$

Given $\beta_m \neq 0$, we have from (31) $e_m^T v_j = 0$ and since T_m is unreduced, $e_m^T v_j \neq 0$, c.f. [17, Theorem 7.9.3].

ii.) To show that $u_j \neq \pm e_{m+1}$, we also argue via contradiction. Let $u_j = \pm e_{m+1}$. Then from (19) and (20) we have,

$$(T_m - \mu_j I_m)v_j = 0 \tag{32}$$

$$\pm\beta_m e_m = \sigma_{\downarrow j} v_j. \tag{33}$$

Given $\beta_m \neq 0$, we have from (33) that $v_j = \pm e_m$. Then from (32), we have $\beta_{m-1} = 0$. Contradiction to T_m being unreduced. □

Theorem 2.4.2. *Let T_m (5) be unreduced and $\beta_m \neq 0$. Given the singular value relationships (19) and (20) with $\mu_j = \theta_j$. Then $z_j \neq \pm x_j$.*

Proof. To show that $z_j \neq \pm x_j$, we argue via contradiction. Let $x_j = \pm z_j$ then $y_j = \pm v_j$. From (23) and Corollary 2.4.1 we have

$$0 = (T_m - \theta_j I_m)y_j = \pm(T_m - \theta_j I_m)v_j = \pm\sigma_{\downarrow j} u_{j(1:m)} \neq 0. \tag{34}$$

□

Therefore, the refined Ritz and Ritz vectors do not coincide until $\sigma_{\downarrow j} = 0$.

Define

$$r_j \;:=\; u_j - ([v_j;0]^T u_j)[v_j;0] \tag{35}$$

and notice that the refined Ritz vector and residual vectors satisfy $z_j^T(P_{m+1}r_j) = 0$. This is a requirement for restarting. Then from (26) we have similar to (9) for $j = 1, \ldots, k$,

$$Az_j \;=\; \rho_j z_j + \sigma_{\downarrow j} P_{m+1} r_j. \tag{36}$$

However, the setup of the thick–restarted method as described in Section 2.3 cannot be used with $\{\rho_j, z_j\}$ in this context, since the residual vectors $\sigma_{\downarrow j} P_{m+1} r_j$ are not multiples of each other for different refined Ritz pairs $\{\rho_j, z_j\}$. This is shown in Theorem 2.4.3. Below is a needed relationship before establishing the result. Notice from (19), (25), and (35) that

$$\begin{bmatrix} T_m - \rho_j I_m \\ \beta_m e_m^T \end{bmatrix} v_j \;=\; \sigma_{\downarrow j} r_j \tag{37}$$

and if $\{\rho_j, v_j\}$ is not an eigenpair of T_m then $\sigma_{\downarrow j}(u_{j(1:m)} - (v_j^T u_{j(1:m)})v_j) \neq 0$.

Theorem 2.4.3. *Given refined Ritz pairs $\{\rho_{j_1}, z_{j_1}\}$ and $\{\rho_{j_2}, z_{j_2}\}$ with $\rho_{j_1} \neq \rho_{j_2}$ satisfying (36) and $\{\rho_{j_1}, v_{j_1}\}$ and $\{\rho_{j_2}, v_{j_2}\}$ not eigenpairs of T_m, we have $\sigma_{\downarrow j_1} P_{m+1} r_{j_1} \neq \gamma \sigma_{\downarrow j_2} P_{m+1} r_{j_2}$ for some scalar γ.*

Proof. To show that $\sigma_{\downarrow j} P_{m+1} r_{j_1} \neq \gamma \sigma_{\downarrow j} P_{m+1} r_{j_2}$ for some scalar γ, we argue via contradiction. Let $\sigma_{\downarrow j} P_{m+1} r_{j_1} = \gamma \sigma_{\downarrow j} P_{m+1} r_{j_2}$ then from multiplying (36) with $j = j_2$ from the left by $z_{j_1}^T$ and then by $z_{j_2}^T$ when $j = j_1$ gives the relationships $v_{j_1}^T T_m v_{j_2} - \rho_{j_2} v_{j_1}^T v_{j_2} = 0$ and $v_{j_2}^T T_m v_{j_1} - \rho_{j_2} v_{j_2}^T v_{j_1} = 0$. Since $\rho_{j_1} \neq \rho_{j_2}$ this implies $v_{j_2}^T T_m v_{j_1} = 0$ and $v_{j_2}^T v_{j_1} = 0$. From (37) we have $(T_m - \rho_{j_1} I_m) v_{j_1} = \gamma (T_m - \rho_{j_2} I_m) v_{j_2} \neq 0$ and $\beta_m e_m^T v_{j_1} = \gamma \beta_m e_m^T v_{2_1}$. Therefore,

$$\begin{aligned} 0 < v_{j_1}^T (T_m - \rho_{j_1} I_m)(T_m - \rho_{j_1} I_m) v_{j_1} &= \gamma v_{j_1}^T (T_m - \rho_{j_1} I_m)(T_m - \rho_{j_2} I_m) v_{j_2} \\ &= \gamma v_{j_1}^T T_m^2 v_{j_2}. \end{aligned}$$

For v_{j_1} from (19) and (20) we have $((T_m - \mu_{j_1} I_m)(T_m - \mu_{j_1} I_m) + \beta_m^2 e_m e_m^T) v_{j_1} = \sigma_{j_1}^2 v_{j_1}$ and after left multiplying by $v_{j_2}^T$ we get $\gamma v_{j_1}^T T_m^2 v_{j_2} = -\beta_m^2 \gamma^2 (e_m^T v_{j_2})^2 < 0$. □

A restarted technique such as setting the starting vector p_1 in Algorithm 2.1 as a linear combination of k desired approximate refined Ritz vectors can be used, see [6]. We propose a different linear combination with constants chosen in a similar fashion outlined in [15]. Before presenting our restarted scheme, we will first focus on adjusting the refined Ritz vectors to reduce the residual norm. We also include some results and motivational remarks.

2.5 Iterative Refined Ritz vectors

Considering the refined Ritz pair $\{\rho_j, z_j\}$ may provide a better approximation by having a "smaller" norm (22) when used together, we propose iteratively refining the approximation. That is, set $\mu_j = \rho_j$ in (17) and re–compute the refined Ritz vectors as stated in (21) with the updated μ_j. This process can be repeated and creates an iterative scheme with a sequence of refined Ritz pairs, $\{\rho_j^{(i)}, z_j^{(i)}\}$ for $i = 1, 2, \ldots$. The process terminates with a refined Ritz pair $\{\hat{\rho}_j, \hat{z}_j\}$ that has favorable properties. We will refer to $\{\hat{\rho}_j, \hat{z}_j\}$, as the iterative refined Ritz value and vector respectively, and collectively as the iterative refined Ritz pair. Algorithm 5.1 outlines the computational process which we follow up with remarks. To show convergence, and hence termination of the iterative scheme, we establish via Theorem 2.5.1 that the nonnegative sequence $\sigma_{\downarrow_j}^{(i)}$ computed from this process is bounded, decreasing, and hence converges. Let $\hat{\sigma}_{\downarrow_j}$ be the value to which the sequence $\sigma_{\downarrow_j}^{(i)}$ converges.

Theorem 2.5.1. *Let $\mu_j = \rho_j^{(i-1)}$ with $\rho_j^{(0)} = \theta_j$, $z_j^{(i)} = P_m v_j^{(i)}$, and $\rho_j^{(i)} = z_j^{(i)T} A z_j^{(i)}$ for $i = 1, 2, \ldots$. Then the computed smallest singular values $\sigma_{\downarrow_j}^{(i)}$ for $i = 1, 2, \ldots$ from the equations (19) and (20) to solve (17) is a nonnegative bounded decreasing sequence and hence converges.*

Proof. We compute

$$\begin{aligned} 0 \leq \sigma_{\downarrow j}^{(i+1)} = \sigma_{\downarrow j}^{(i+1)} \|u_j^{(i+1)}\| &= \|\sigma_{\downarrow j}^{(i+1)} u_j^{(i+1)}\| \\ &= \|(T_{m+1,m} - \rho_j^{(i)} I_{m+1,m}) v_j^{(i+1)}\| \\ &\leq \|(T_{m+1,m} - \rho_j^{(i)} I_{m+1,m}) v_j^{(i)}\|. \end{aligned} \tag{38}$$

We have the inequality in (38) since $v_j^{(i+1)}$ satisfies the minimization property in (21) when $\mu_j = \rho_j^{(i)}$. Note that $v_j^{(i)}$ satisfies the minimization property in (21) when $\mu_j = \rho_j^{(i-1)}$, therefore we have

$$\begin{aligned} (T_{m+1,m} - \rho_j^{(i-1)} I_{m+1,m}) v_j^{(i)} &= \sigma_{\downarrow j}^{(i)}\ u_j^{(i)} \\ T_{m+1,m} v_j^{(i)} &= \sigma_{\downarrow j}^{(i)}\ u_j^{(i)} + \rho_j^{(i-1)} I_{m+1,m} v_j^{(i)}. \end{aligned} \tag{39}$$

From (25) we have

$$\rho_j^{(i)} - \rho_j^{(i-1)} \ = \ \sigma_{\downarrow j}^{(i)}\ v_j^{(i)^T} u_{j(1:m)}{}^{(i)}. \tag{40}$$

Plugging (39) and (40) into (38) and continuing (38) we have

$$\begin{aligned} 0 \leq \sigma_{\downarrow j}^{(i+1)} &\leq \|T_{m+1,m} v_j^{(i)} - \rho_j^{(i)} I_{m+1,m} v_j^{(i)}\| \\ &= \sigma_{\downarrow j}^{(i)}\ \|u_j^{(i)} - ([v_j; 0]^{(i)^T} u_j^{(i)})[v_j; 0]^{(i)}\| \\ &\leq \sigma_{\downarrow j}^{(i)}\ . \end{aligned} \tag{41}$$

The last inequality in (41) comes from using Lemma 2.8.1. A strict inequality exists if $([v_j; 0]^{(i)^T} u_j^{(i)}) \neq 0$, see Lemma 2.8.1. □

Notice from (27) and (41) we have

$$\begin{aligned} \|Az_j^{(i+1)} - \rho_j^{(i+1)} z_j^{(i+1)}\| \ &\leq \ \sigma_{\downarrow j}^{(i+1)} \\ &\leq \ \sigma_{\downarrow j}^{(i)}\ \|u_j^{(i)} - ([v_j; 0]^{(i)^T} u_j^{(i)})[v_j; 0]^{(i)}\| \\ &= \ \|Az_j^{(i)} - \rho_j^{(i)} z_j^{(i)}\| \end{aligned} \tag{42}$$

which implies $\{\rho_j^{(i+1)}, z_j^{(i+1)}\}$ can be a better approximation than $\{\rho_j^{(i)}, z_j^{(i)}\}$. If $\sigma_{\downarrow_j}^{(i)} = 0$ for some i we have from (42) that $\{\rho_j^{(i)}, z_j^{(i)}\}$ is an exact eigenpair of A. We assume for the remainder of the section that $0 < \hat{\sigma}_{\downarrow_j} \leq \sigma_{\downarrow_j}^{(i)}$.

Another point of view of the equations (19) and (20) is as an eigenvalue problem. In this context, define $H(\mu_j) := (T_m - \mu_j I_m)(T_m - \mu_j I_m) + \beta_m^2 e_m e_m^T$ where $\mu_j = \rho_j = v_j^T T_m v_j$ is a function of the unit vector v_j. Therefore, we have

$$H(v_j)v_j \quad = \quad \sigma_{\downarrow_j}^2 v_j \tag{43}$$

and we are searching for the smallest eigenvalue $\sigma_{\downarrow_j}^2$ and associated unit eigenvector v_j. The vector u_j can be obtained by

$$u_j \quad = \quad 1/\sigma_{\downarrow_j}(T_{m+1,m} - \mu_j I_{m+1,m})v_j. \tag{44}$$

Equation (43) is referred to as an eigenvector–dependent nonlinear eigenvalue problem (NEPv) and the most commonly used routine for solving (43) is the simple self-consistent field (SCF) iteration process, see [3] and reference within.

The computation of iterative refined values is outlined in the Iterative Refined Algorithm 5.1. We assume $m \ll n$ and the computational time required per iteration for Algorithm 5.1 is negligible in comparison to the computational time required for a matrix–vector product with A when n is very large.

Algorithm 5.1 Iterative Refined

1: **Input:** $T_{m+1,m} \in \mathbb{R}^{m+1 \times m}$ (18) and $\{\mu_j\}_{j=1}^k$,
2: **Output:** $\{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_{\downarrow_j}\}_{j=1}^k$
3: **for** $j = 1, 2, \ldots, k$ **do**
4: **for** $i = 1, 2, \ldots, maxit$ **do**
5: Compute $v_j^{(i)}, u_j^{(i)}$, and $\sigma_{\downarrow_j}^{(i)}$ using either (19) and (20) or (43) and (44)
6: Set $\rho_j^{(i)} := v_j^{(i)^T} T_m v_j^{(i)}$
7: Check convergence
8: Set $\mu_j := \rho_j^{(i)}$

There are several options for step 7 in Algorithm 5.1 on checking convergence. For example, convergence can be checked by $|\rho_j^{(i)} - \rho_j^{(i-1)}|/|\rho_j^{(i)}| < eps$, where eps

is machine epsilon. Some heuristics for stopping are provided in [3] and references within when using the SCF iteration to solve NEPv (43). It should be noted that stagnation can occur while using finite arithmetic and we propose including a check to exit when detected to avoid unnecessary iterations. As the restarted hybrid method presented in section 2.6 converges we notice via numerical experiments the number of iterations in Algorithm 5.1 reduce quickly to only a handful.

We see from Theorem 2.5.1 and the relationship $\sigma_{\downarrow j}^2 = v_j^T H(v_j) v_j$ that the output $\{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_{\downarrow j}\}$ from Algorithm 5.1 satisfies,

$$(T_{m+1,m} - \hat{\rho}_j I_{m+1,m})\hat{v}_j = \hat{\sigma}_{\downarrow j}\hat{u}_j \tag{45}$$

$$(T_{m+1,m} - \hat{\rho}_j I_{m+1,m})^T \hat{u}_j = \hat{\sigma}_{\downarrow j}\hat{v}_j \tag{46}$$

where $\hat{\rho}_j = \hat{v}_j^T T_m \hat{v}_j$. Equate the first m rows of (45) and left multiply by $\hat{v}_j^T$ to get

$$\hat{\sigma}_{\downarrow j}\hat{v}_j^T \hat{u}_{j(1:m)} = 0. \tag{47}$$

Using (35), (36), and (47) we have

$$A\hat{z}_j = \hat{\rho}_j \hat{z}_j + \hat{\sigma}_{\downarrow j} P_{m+1}\hat{u}_j \tag{48}$$

where $\hat{z}_j = P_m \hat{v}_j$ and $\hat{z}_j^T P_{m+1}\hat{u}_j = 0$. Notice from (22), (42), and (48)

$$\hat{\sigma}_{\downarrow j} = \|A\hat{z}_j - \hat{\rho}_j \hat{z}_j\| \leq \|Az_j - \rho_j z_j\| \leq \|Az_j - \theta_j z_j\| \leq \|Ax_j - \theta_j x_j\|. \tag{49}$$

Notice that in floating point arithmetic we have $\hat{\sigma}_{\downarrow j}\hat{v}_j^T \hat{u}_{j(1:m)} \approx 0$ and should be included in equation (48) when used in computer codes, c.f. (36). It is not included in establishing subsequent results and equations, i.e. we assume (47) holds.

Although the results listed in [9, 11, 18] were developed for refined Ritz and Ritz pairs when $\mu_j = \theta_j$, we see from the relationships (42) and (49) that similar results apply for iterative refined Ritz. In particular, we have when $\hat{\sigma}_{\downarrow j} = 0$, vectors $\hat{z}_j$ and x_j are parallel to an eigenvector of A and $\hat{\rho}_j = \theta_j$ is an exact eigenvalue of A. Notice

that Corollary 2.4.1 does not depend on $\mu_j = \theta_j$, however Theorem 2.4.2 depended on setting $\mu_j = \theta_j$, therefore we state the result in this context for $\hat{z}_j$.

Theorem 2.5.2. *Let T_m (5) be unreduced and $\beta_m \neq 0$. Given the singular value relationships (45) and (46), then $\hat{z}_j \neq \pm x_j$.*

Proof. To show that $\hat{z}_j \neq \pm x_j$, we argue via contradiction. Let $x_j = \pm\hat{z}_j$ then $y_j = \pm\hat{v}_j$ and $\hat{\rho}_j = \hat{v}_j^T T_m \hat{v}_j = y_j^T T_m y_j = \theta_j$. From (45) and Corollary 2.4.1 we have

$$0 = (T_m - \theta_j I_m) y_j \quad = \quad \pm(T_m - \hat{\rho}_j I_m)\hat{v}_j = \pm\hat{\sigma}_{\downarrow j}\hat{u}_{j(1:m)} \neq 0 \tag{50}$$

□

Therefore, the iterative refined Ritz and Ritz vectors do not coincide unless $\hat{\sigma}_{\downarrow j} = 0$. Theorem 2.4.3 also does not depend on $\mu_j = \theta_j$ and the result holds in this context. That is, the set-up of the thick–restarted method as described in Section 2.3 cannot be used with $\{\hat{\rho}_j, \hat{z}_j\}$ in this context, since the residual vectors $\sigma_{\downarrow j} P_{m+1}\hat{r}_j$ are not multiples of each other for different iterative refined pairs $\{\hat{\rho}_j, \hat{z}_j\}$. The iterative refined Ritz pair have a "smaller" residual norm and possess similar properties to the refined Ritz pair. The following examples provide motivation on our restarting technique.

Example 2.5.1 Let A be the 4×4 symmetric matrix,

$$A = \begin{bmatrix} 9 & 1 & -2 & 1 \\ 1 & 8 & -3 & -2 \\ -2 & -3 & 7 & -1 \\ 1 & -2 & -1 & 6 \end{bmatrix}. \tag{51}$$

The eigenvalues of A are 12, 9, 6, 3. Using MLan(0) Algorithm 2.1 with $p_1 = \frac{1}{2}[1\ 1\ 1\ 1]^T$ and $m = 3$, gives a tridiagonal matrix T_3 with eigenvalues $\theta_1 = 11.7913$, $\theta_2 = 7.4755, \theta_3 = 3.0239$, and $\beta_3 = 1.8035$. We have for the largest eigenpair,

$$\hat{\sigma}_{\downarrow 1} = 0.831397 < \sigma_{\downarrow 1}\|r_1\| = 0.831400 \quad < \quad \beta_3 |e_3^T y_1| = 0.885392 \tag{52}$$

where $\mu_1 = \theta_1$ in Algorithm 5.1. Equation (52) shows the residual norm with iterative refined pair is "smaller", with similar results for the other eigenpairs. In practice, the matrix A is very large and restarting is required. A restarted Lanczos method depends on many things for successful complementation, one of which is a "good" (re)starting vector. For a fair comparison, Table 1 displays the Ritz residual norms $\beta_3|e_3^T y_1|$ associated with $\{\theta_1, y_1\}$ for A where we set the (re)starting vector p_1 in MLan(0) Algorithm 2.1 on the next restart to be the computed iterative refined Ritz $P_3\hat{v}_1$, refined Ritz $P_3 v_1$, and Ritz vector $P_3 y_1$. Also, included in Table 1 is the sine of the angle for each vector $P_3\hat{v}_1$, $P_3 v_1$, and $P_3 y_1$ with the desired eigenvector x associated with the largest eigenvalue 12 of the matrix A. It should be noted that Krylov subspaces associated with each column in Table 1 are different, since they depend on the starting vector. However, we do see from Table 1 that using the iterative refined Ritz vector $P_3\hat{v}_1$ we are able to obtain a smaller Ritz residual norm on each restart and a starting vector "closer" to the desired eigenvector. Although the results for iterative refined Ritz norms and angles may appear to be only marginally smaller than refined Ritz norms, in a restarted scheme for large matrices this can be significant.

Table 1. Example 2.5.1. Displays the Ritz residual norms $\beta_3|e_3^T y_1|$ (7) associated with $\{\theta_1, y_1\}$ for different (re)starting vector p_1 in MLan(0) Algorithm 2.1 on the next restart. Also, displays the sine of the angle for each vector with the eigenvector x associated with the largest eigenvalue 12.

Restart	Iterative refined Ritz		Refined Ritz		Ritz	
	$\beta_3\|e_3^T y_1\|$	$\sin\angle(x, P_3\hat{v}_1)$	$\beta_3\|e_3^T y_1\|$	$\sin\angle(x, P_3 v_1)$	$\beta_3\|e_3^T y_1\|$	$\sin\angle(x, P_3 y_1)$
1	$2.4589 \cdot 10^{-2}$	$2.8770 \cdot 10^{-3}$	$2.4820 \cdot 10^{-2}$	$2.9060 \cdot 10^{-3}$	$6.6286 \cdot 10^{-2}$	$7.8691 \cdot 10^{-3}$
2	$7.0237 \cdot 10^{-4}$	$2.0977 \cdot 10^{-4}$	$7.1591 \cdot 10^{-4}$	$2.1353 \cdot 10^{-4}$	$2.1557 \cdot 10^{-3}$	$5.7290 \cdot 10^{-4}$
3	$1.8391 \cdot 10^{-5}$	$2.1518 \cdot 10^{-6}$	$1.8918 \cdot 10^{-5}$	$2.2148 \cdot 10^{-6}$	$1.5244 \cdot 10^{-4}$	$1.8096 \cdot 10^{-5}$
4	$5.2531 \cdot 10^{-7}$	$1.5699 \cdot 10^{-7}$	$5.4567 \cdot 10^{-7}$	$1.6187 \cdot 10^{-7}$	$4.9572 \cdot 10^{-6}$	$1.3167 \cdot 10^{-6}$

Example 2.5.1 is a good illustrative example that shows that we can get better results with iterative refined Ritz vectors. However, this example's features are ideal

with well–separated eigenvalues of A and large T_m ($m = 3$ compared with $n = 4$) that yields a good approximation to all eigenpairs on the first iteration. In practice, A will be very large, $m \ll n$, and the corresponding generated Krylov subspace yielding a poor approximation to the desired eigenpairs. When m is kept very small we experienced very poor results, often with stagnation. This can be contributed in part to an overall poor approximation from the Krylov subspace and the computational process of refined or iterative refined Ritz values. The following example illustrates this undesirable scenario.

Example 2.5.2 Let A be the 3×3 symmetric matrix,

$$A = \begin{bmatrix} 0.0025 & 0.0485 & 0 \\ 0.0485 & 2.1509 & 2.3 \\ 0 & 2.3 & 0 \end{bmatrix}. \tag{53}$$

The eigenvalues of A are 3.6149, $2.4989 \cdot 10^{-3}$, and -1.4640. Using MLan(0) Algorithm 2.1 on matrix A with $p_1 = [1\ 0\ 0]^T$ and $m = 2$, yields a matrix $T_{3,2}$ that consist of the first 2 columns of A, with eigenvalues of $T_{3,2}(1\colon 2, 1\colon 2)$ as $\theta_1 = 2.1520$ and $\theta_2 = 0.0014$. We used Algorithm 5.1 with $\mu_1 = \theta_1 = 2.1520$ for computing refined Ritz and iterative refined Ritz pairs. Table 2a below displays the output of the refined Ritz ρ_1 and iterative refined $\hat{\rho}_1$ values and the sine of the angles between the eigenvector x associated with the largest eigenvalue 3.6149, and the eigenvectors y_1 and y_2, associated with θ_1 and θ_2, respectively. We see that refined Ritz pair $\{\rho_1, v_1\}$ and iterative refined Ritz pair $\{\hat{\rho}_1, \hat{v}_1\}$ are "closer" to $\{\theta_2, y_2\}$ than $\{\theta_1, y_1\}$ even though approximations set $\mu_1 = \theta_1$. This can cause a restarted method with refined Ritz or iterative refined Ritz to stagnate or converge slowly. See Table 2b where we set the (re)starting vector p_1 in MLan(0) Algorithm 2.1 on the next restart to be the computed iterative refined Ritz $P_2\hat{v}_1$, refined Ritz $P_2 v_1$, and Ritz vector $P_2 y_1$. The initial iterative refined Ritz pair is very close to the undesired value $\{\theta_2, y_2\}$ and the process cannot recover, causing stagnation. The initial refined Ritz pair is not as close, and does recover, however the overall convergence is a lot slower than

using Ritz vector to restart. Also, the computed refined Ritz residual $\sigma_{\downarrow 1}\|r_1\|$ for the method that restarts with refined Ritz vector was only marginally better, e.g. on restart 4 with refined Ritz vector method we have $\sigma_{\downarrow 1}\|r_1\| = 5.3081 \cdot 10^{-5}$ (compared to $\beta_2|e_2^T y_1| = 5.3121 \cdot 10^{-5}$). In this example, notice that the initial Ritz vector is significantly closer to the desired eigenvector x and restarting with Ritz vector converges a lot faster.

Table 2. Example 2.5.2.

Iterative Refined Ritz $\hat{\rho}_1 = 0.0017$			Refined Ritz $\rho_1 = 0.0655$			Ritz
$\sin\angle(x, P_2\hat{v}_1)$	$\sin\angle(y_1, \hat{v}_1)$	$\sin\angle(y_2, \hat{v}_1)$	$\sin\angle(x, P_2 v_1)$	$\sin\angle(y_1, v_1)$	$\sin\angle(y_2, v_1)$	$\sin\angle(x, P_2 y_1)$
1.000	0.0120	0.9999	0.9904	0.1727	0.9850	0.5368

(a) Display results when using Algorithm 5.1 with $\mu_1 = \theta_1 = 2.1520$ applied to $T_{3,2}$. Also, displays the sine of the angle for each vector with the eigenvector x associated with the largest eigenvalue 3.6149.

Restart	Iterative refined Ritz		Refined Ritz		Ritz	
	$\beta_2\|e_2^T y_1\|$	$\sin\angle(x, P_2\hat{v}_1)$	$\beta_2\|e_2^T y_1\|$	$\sin\angle(x, P_2 v_1)$	$\beta_2\|e_2^T y_1\|$	$\sin\angle(x, P_2 y_1)$
1	$1.2276 \cdot 10^{0}$	$9.9998 \cdot 10^{-1}$	$9.0575 \cdot 10^{-1}$	$1.8281 \cdot 10^{-1}$	$2.7847 \cdot 10^{-2}$	$7.7072 \cdot 10^{-3}$
2	$2.2985 \cdot 10^{0}$	$1.0000 \cdot 10^{0}$	$3.5993 \cdot 10^{-2}$	$9.9344 \cdot 10^{-3}$	$3.3735 \cdot 10^{-4}$	$6.6449 \cdot 10^{-5}$
3	$1.2275 \cdot 10^{0}$	$1.0000 \cdot 10^{0}$	$1.3828 \cdot 10^{-3}$	$2.7442 \cdot 10^{-4}$	$2.9083 \cdot 10^{-6}$	$8.0466 \cdot 10^{-7}$
4	$2.2983 \cdot 10^{0}$	$1.0000 \cdot 10^{0}$	$5.3121 \cdot 10^{-5}$	$1.4661 \cdot 10^{-5}$	$3.5229 \cdot 10^{-8}$	0

(b) Displays the Ritz residual norms $\beta_2|e_2^T y_1|$ (7) associated with $\{\theta_1, y_1\}$ for different (re)starting vector p_1 in MLan(0) Algorithm 2.1 on the next restart. Also, displays the sine of the angle for each vector with the eigenvector x associated with the largest eigenvalue 3.6149.

Although the example 2.5.2 is contrived, it illustrates what can happen. The next example, further illustrates the problem.

Example 2.5.3 Let $A = \text{diag}(1{:}500)$ be a 500×500 diagonal matrix. We are searching for the largest eigenpair. We set $m = 2$ and the (re)starting vector p_1 in MLan(0) Algorithm 2.1 on the next restart to be the computed Ritz vector $P_2 y_1$, refined Ritz $P_2 v_1$, or iterative refined Ritz $P_2\hat{v}_1$. We ran the example 100 times with a different beginning random vector p_1. The results are presented in Figure 2a, Figure 2b, and Figure 2c and show that restarting with iterative refined Ritz $P_2\hat{v}_1$ or refined

Ritz $P_2 v_1$ caused stagnation, or erratic behavior and slow convergence. Restarting with Ritz vector, Figure 2a was not erratic, but convergence was slow.

Example 2.5.1 showed that when Krylov subspace was a fairly "good" subspace that restarting with iterative refined Ritz vector provided the better results. However, examples 2.5.2 and 2.5.3 demonstrated the pitfalls of using a conventional restarting scheme with just (iterative) refined Ritz vectors when the Krylov subspace was a "poor" subspace. Although examples 2.5.2 and 2.5.3 may show that the iterative refined Ritz vectors to be problematic, the iterative refined Ritz vectors highlight, more so than refined Ritz vectors, when they should not be used to restart. Using this information and that restarting with Ritz vectors convergence was not erratic (although slow), we developed a hybrid method that switches, depending on some parameters, between thick–restarting with Ritz vectors and restarting with iterative refined Ritz vectors and show this combination overcomes the stagnation and erratic behavior producing a faster overall converging method.

2.6 Hybrid Methods

The hybrid method developed here uses thick–restarted as the main routine and, under certain conditions, switches to restarting with iterative refined Ritz vectors. The iterative refined Ritz vectors from a "good" Krylov subspace can be better approximations, but a trade off is a restarted scheme that loses the benefits of thick–restarted which are crucial and traced back to the IRL method. Using only the thick–restarted with a small m value converges very slowly or not at all. We have found that when the Krylov subspace is "good" that switching to restarting with iterative refined Ritz vectors, even for a few iterations, results in a faster overall convergence.

For $k = 1$ (single eigenpair) restating equations (9) and (48) for the Ritz pair

$\{x_1, \theta_1\}$ and the iterative refined Ritz pair $\{\hat{z}_1, \hat{\rho}_1\}$

$$Ax_1 = [x_1, p_2] \begin{bmatrix} \theta_1 \\ \bar{\beta}_1 \end{bmatrix} \quad \text{where } p_2 = f/\beta_m \tag{54}$$

$$A\hat{z}_1 = [\hat{z}_1, p_2] \begin{bmatrix} \hat{\rho}_1 \\ \hat{\sigma}_1 \end{bmatrix} \quad \text{where } p_2 = P_{m+1}\hat{u}_1. \tag{55}$$

Depending on certain parameters for switching described in Section 2.6.1, we can restart by calling the MLan(1) Algorithm 2.1 with starting vector p_2. This is slightly different than a restarted Lanczos method, by using relationships in equations (54) and (55) to avoid a matrix–vector product with A on each restart.

The hybrid method for finding $k > 1$ eigenpairs has added challenges. The thick–restarted algorithm is set up to compute $k \geq 1$ eigenpairs, however the iterative refined Ritz does not fit this structure. Therefore, we implement a standard restart technique with a starting vector p_1 constructed as a linear combination of k iterative refined Ritz vectors

$$p_1 = \sum_{j=1}^{k} c_j \hat{z}_j. \tag{56}$$

This was the set up for the refined Ritz algorithm [6, Algorithm 1] where the combination of the refined Ritz vectors for the starting vectors p_1 were constructed based on coefficients c_j chosen as described in Saad [19]. In our development, we chose the coefficients c_j in a similar way to the description outlined by Morgan [15]. In simplest terms, for Ritz vectors x_j, constants c_j are chosen for a starting vector $p_1 = c_1 x_1 + \ldots + c_k x_k$ to eliminate the coefficients $\beta_m e_m^T y_j$ multiplying the common residual vector p_{m+1}. That is, the choice of coefficients removes p_{m+1} when the starting vector p_1 is multiplied by A in the next iteration to build out the Krylov subspace. The coefficients can be determined by solving a certain homogenous $k - 1 \times k$ linear system. It was then proven by Morgan with these specially chosen coefficients c_j that the span$\{p_1, Ap_1, \ldots, A^{k-1}p_1\}$ = span$\{x_1, x_2, \ldots, x_k\}$ which is the same subspace resulting from implementing Sorensen's IRA method [22]. We refer the reader

to [15] for specific details and theoretical results. Building on this idea, we can, in a similar way, remove some of the coefficients associated with p_{m+1} for iterative refined vectors. The rationale on implementing this technique is to have a restarted method that may inherit similar convergence benefits of the IRL method. We did observe fast convergence, even with removing only some of the coefficients associated with p_{m+1}.

From equations (24), (48), and (56) we have

$$Ap_1 \;=\; \sum_{j=1}^{k} c_j(\hat{\rho}_j P_m \hat{v}_j + \beta_m e_m^T \hat{v}_j p_{m+1} + \hat{\sigma}_{\downarrow j} P_m \hat{u}_{j(1:m)}). \tag{57}$$

Therefore, we select coefficients c_j such that $\beta_m e_m^T \hat{v}_j c_j = 0$ which removes p_{m+1} from (57). If we ignore $\hat{\sigma}_{\downarrow j} P_m \hat{u}_{j(1:m)}$ in (57) and all future occurrences, as we multiple (57) by A we obtain a similar $k-1 \times k$ homogenous system linear system to the one presented in [15, Section 3] where i^{th} row is represented as,

$$\beta_m \left[\begin{array}{cccc} \hat{\rho}_1^{i-1} e_m^T \hat{v}_1 & \hat{\rho}_2^{i-1} e_m^T \hat{v}_2 & \ldots & \hat{\rho}_k^{i-1} e_m^T \hat{v}_k \end{array} \right]. \tag{58}$$

If we leave $\hat{\sigma}_{\downarrow j} P_m \hat{u}_{j(1:m)}$ in (57) and continue with p_{m+1} removed in (57), we have,

$$A^2 p_1 \;=\; \sum_{j=1}^{k} c_j(\hat{\rho}_j^2 P_m \hat{v}_j + \hat{\rho}_j \hat{\sigma}_{\downarrow j} P_m \hat{u}_{j(1:m)} + \beta_m \hat{\rho}_j e_m^T \hat{v}_j p_{m+1} + \hat{\sigma}_{\downarrow j}\; A P_m \hat{u}_{j(1:m)}) \tag{59}$$

where

$$\hat{\sigma}_{\downarrow j}\; A P_m \hat{u}_{j(1:m)} \;=\; \hat{\sigma}_{\downarrow j} P_m T_m \hat{u}_{j(1:m)} + \beta_m (e_m^T T_m \hat{v}_j - \hat{\rho}_j e_m^T \hat{v}_j) p_{m+1}. \tag{60}$$

Collecting terms multiplying p_{m+1} in (59) and (60) we have,

$$(\beta_m \hat{\rho}_j e_m^T \hat{v}_j + \beta_m (e_m^T T_m \hat{v}_j - \hat{\rho}_j e_m^T \hat{v}_j)) c_j \;=\; \beta_m e_m^T T_m \hat{v}_j c_j. \tag{61}$$

We therefore select coefficients c_j such that $\beta_m e_m^T T_m \hat{v}_j c_j = 0$ which removes p_{m+1} from (59). If we ignore $\hat{\sigma}_{\downarrow j} P_m T_m \hat{u}_{j(1:m)}$ in (60) and all future occurrences, as we multiply (57) by A we obtain the following $k-1 \times k$ homogenous system linear system where coefficient matrix has the same first row as in (58) and is represented as,

$$\beta_m \left[\begin{array}{cccc} e_m^T \hat{v}_1 & e_m^T \hat{v}_2 & \ldots & e_m^T \hat{v}_k \\ \hat{\rho}_1^{i-2} e_m^T T_m \hat{v}_1 & \hat{\rho}_2^{i-2} e_m^T T_m \hat{v}_2 & \ldots & \hat{\rho}_k^{i-2} e_m^T T_m \hat{v}_k \end{array} \right] \;\; i > 1. \tag{62}$$

Notice from (45) that as $\hat{\sigma}_{\downarrow j}$ approaches zero, we expect $T_m \hat{v}_j$ to approach $\hat{\rho}_j \hat{v}_j$ and (62) would become like the homogenous system (58). If we leave $\hat{\sigma}_{\downarrow j} P_m T_m \hat{u}_{j(1:m)}$ in (60) and continue to remove p_{m+1} we get the $k-1 \times k$ homogenous linear system where coefficient matrix has the same first two rows as in (62) and is represented as,

$$\beta_m \begin{bmatrix} e_m^T \hat{v}_1 & e_m^T \hat{v}_2 & \dots & e_m^T \hat{v}_k \\ e_m^T T_m \hat{v}_1 & e_m^T T_m \hat{v}_2 & \dots & e_m^T T_m \hat{v}_k \\ \hat{\rho}_1^{i-2} e_m^T T_m \hat{v}_1 + s_1 & \hat{\rho}_2^{i-2} e_m^T T_m \hat{v}_2 + s_2 & \dots & \hat{\rho}_k^{i-2} e_m^T T_m \hat{v}_k + s_k \end{bmatrix} \quad i > 2 \tag{63}$$

where

$$s_j \;=\; \hat{\sigma}_{\downarrow j} \sum_{\ell=3}^{i} \hat{\rho}_j^{i-\ell} e_m^T T_m^{\ell-2} \hat{u}_{j(1:m)} \qquad 1 \le j \le k. \tag{64}$$

Likewise, as $\hat{\sigma}_{\downarrow j}$ approaches zero, we expect $T_m \hat{v}_j$ to approach $\hat{\rho}_j \hat{v}_j$ and s_j to approach zero, hence (63) would also become like the homogenous system (58). The matrices become more complicated and ill–conditioned as we include more terms for eliminating the coefficients multiplying p_{m+1}. We do assume that k is small and have observed similar results with all three systems, but more consistent results over a wide range of problems when using (62) or (63). We solved the $k-1 \times k$ homogenous linear system by finding the null space vector using the singular value decomposition. When a column becomes numerically zero, indicating an iterative refined Ritz vector has converged, we remove that column and the last row of the matrix and compute the null space vector of the reduced matrix. We then replace the corresponding coefficient with the norm of the iterative refined Ritz residual vector before creating the linear combination for restart.

Notice that when restarting with the linear combination of iterative refined Ritz vectors, a single matrix–vector product can be saved per iteration by utilizing the relationship (57) before restarting. Setting $p_1 = p_1/\|p_1\|$ and setting $\tilde{f}$ to be right side of the equality in (57) multiplied by $1/\|p_1\|$ we have $Ap_1 = \tilde{f}$, $\tilde{\alpha}_1 = p_1^T \tilde{f}$ and $\tilde{f} = \tilde{f} - p_1 \tilde{\alpha}_1$, $\tilde{\beta}_1 = \|\tilde{f}\|$ and

$$Ap_1 \;=\; [p_1, p_2] \begin{bmatrix} \tilde{\alpha}_1 \\ \tilde{\beta}_1 \end{bmatrix} \tag{65}$$

where $p_2 = \tilde{f}/\tilde{\beta}_1$. The MLan(1) Algorithm 2.1 can be continued with p_2. Algorithm 6.1 outlines the hybrid method.

2.6.1 Hybrid Thick–Restarted and Iterative Refined Ritz Algorithm

Hybrid Thick–Restarted and Iterative Refined Ritz Algorithm 6.1 presents the main algorithm of the paper. Algorithm 6.1 starts with the efficient thick–restarted routine. We provide parameters as to when restarting with iterative refined Ritz vectors can be used. The parameters were chosen from numerous experiments on a variety of problems. A careful balance is needed, since the iterative refined Ritz vectors can give a better approximation when the Krylov subspace is "good", but thick–restarted is a more efficient restarting scheme, but often has slower convergence. Also, as illustrated in Example 5.2 in Section 2.5 , the iterative refined Ritz pair may be "closer" to a different Ritz pair of T_m than the originally sought after Ritz pair. Therefore, since m is kept small relative to k we suggest using thick–restarted for the beginning iterations to build a more accurate approximation subspace, i.e. until $\max_{1\le j\le k}|\bar{\beta}_j| \le \epsilon^{0.1}\|A\|$ where ϵ is a user input tolerance for overall convergence and $\bar{\beta}_j$ is from (9). We then check the angle via inner product between $\hat{z}_j$ and x_j, i.e. between $\hat{v}_j$ and y_j. If the angle is acceptable we use iterative refined Ritz vector(s) to restart. Numerous experiments suggest using $\min_{1\le j\le k}|y_j^T\hat{v}_j| > 0.9$. However, when $k > 1$ these conditions alone do not always prevent missing eigenvalues. One solution is to also require the input value μ_j into Iterative Refined Algorithm 5.1 to be the best approximation eigenvalue of A over all computed θ_j's values thus far and to reject using restarting with iterative refined Ritz vectors if the current computed $\hat{\rho}_j$ are not "better" than the past iteration's best approximation. For example, during a current iteration (iter) of Algorithm 6.1, if searching for the k largest in magnitude eigenpairs, we require in step 5 for the call to Algorithm 5.1 that

$$\mu_j \;=\; \max_{1\,\le\,\mathrm{i}\,\le\,\mathrm{iter}} |\theta_j^{(i)}| \quad \text{for} \quad 1 \le j \le k \tag{66}$$

and for step 7

$$|\hat{\rho}_j^{(\text{iter})}| \;\; \geq \;\; \max_{1 \leq \text{i} \leq \text{iter-1}} |\theta_j^{(i)}| \quad \text{for} \quad 1 \leq j \leq k. \tag{67}$$

Similar requirements are made for other desired extreme eigenvalue locations. When $k = 1$ we found that using (66) was a needed requirement for the best results, but encountered poor convergence results when enforcing (67) with $m = 2$, see Examples 7.4 and 7.5 Section 2.7. The following example illustrates the methods presented.

Algorithm 6.1 Hybrid Thick–Restarted and Iterative Refined Ritz

Input: $A \in \mathbb{R}^{n \times n}$: symmetric matrix,
$p_1 \in \mathbb{R}^n$: orthonormal vector,
ϵ: usr specified tolerance,
δ_1: usr specified tolerance on when switching can start
(recommended: $\delta_1 := \epsilon^{0.1}$),
δ_2: usr specified tolerance on when vectors are "close"
(recommended: $\delta_2 := 0.9$),
k: number of desired eigenpairs of A,
m: maximum number of Lanczos vectors.
Output: k approximate desired eigenpair(s) of A.
Compute factorization (3) by MLan(0) Algorithm 2.1
Compute the k desired eigenpair $\{\theta_j, y_j\}_{j=1}^k$ of T_m (5) or $\bar{T}_m$ (16)
Compute $\{\hat{\rho}_j, \hat{v}_j, \hat{u}_j, \hat{\sigma}_j\}_{j=1}^k$ by Iterative Refined Algorithm 5.1 with e.g. μ_j (66)
Check convergence (8) or (68)
if all $\hat{\rho}_j$ converged in Algorithm 5.1 and satisfy e.g. (67) **then**
    if $\max_{1 \leq j \leq k} |\bar{\beta}_j| \leq \delta_1 \|A\|$ and $\min_{1 \leq j \leq k} |x_j^T \hat{z}_j| > \delta_2$ **then**
        if $k > 1$ **then** compute c_j from (58), (62) or (63)
        Restart with iterative refined (55) or (65) via MLan(1) Algorithm 2.1
    else
        Restart with Ritz (54) or (11) via MLan(k) Algorithm 2.1
Goto 4

Example 2.6.1 Let A be the 256,000 $\times$ 256,000 matrix *Lin* from the SuiteSparse Matrix Collection [4]. The largest in magnitude eigenvalue is 1063.63 and the next three largest in magnitude are 1063.32, 1063.31, and 1062.98. We set $m = 15$ and $k = 1$ and $k = 4$. We compared Thick–Restarted Ritz Algorithm 3.1 with Algorithm 6.1. For $k = 1$ we also included the restarted refined Ritz as described in [6, Algorithm

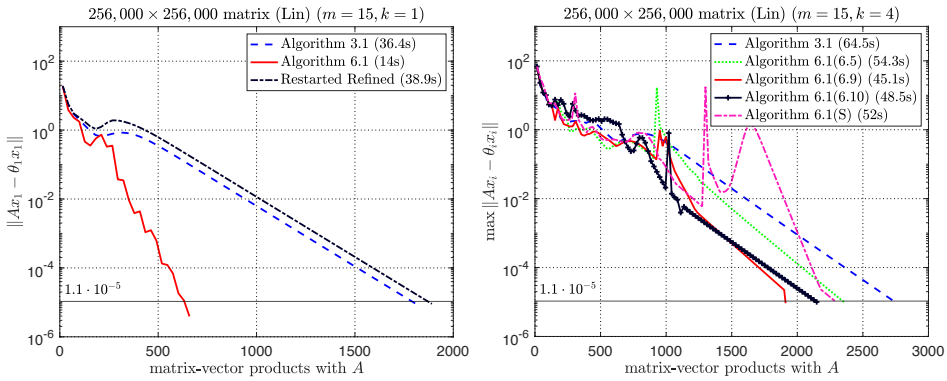


Figure 1. Example 2.6.1. Matrix A is the $256,000 \times 256,000$ matrix *Lin* from the SuiteSparse Matrix Collection [4]. Algorithm 6.1 converges faster than the other restarted methods.

1]. We did not include restarted refined Ritz for $k = 4$ as the method did not converge within 3000 matrix-vector products. We used the same starting vector for each routine, a normalized random vector and a stopping criteria $10^{-8}\|A\|$. For fair comparisons, we recorded the Ritz norm residual $\|Ax_1 - \theta_1 x_1\|$ for $k = 1$ and $\max \|Ax_i - \theta_i x_i\|$ for $k = 4$. For $k = 4$ we compared results using Algorithm 3.1 with Algorithm 6.1 using (58), (62), and (63). We also computed the coefficients in (56) using Saad's method [19] indicated by Algorithm 6.1 (S) in the legend. The graphs in Figure 1 show that Algorithm 6.1 with (62) and (63) outperforms the other routines. Numerical comparison with this matrix with other routines is given in Example 2.7.3 in Section 2.7.

Example 2.6.2 Revisiting Example 5.3 where $A = \text{diag}(1\!:\!500)$ a 500×500 diagonal matrix and $m = 2$. Figures 2a, 2b, and 2c showed for 100 restarts we had stagnation, or erratic behavior and no convergence within 500 matrix-vector products using Ritz vector $P_2 y_1$, refined Ritz $P_2 v_1$ or iterative refined Ritz $P_2 \hat{v}_1$ as the restarting vector in MLan(0) Algorithm 2.1. Figure 2e, Algorithm 6.1, with 100 random restarts always converged within tolerance $10^{-8}\|A\|$ with no more than 170 matrix-vector products and typical convergence between 100 and 150 matrix-vector products. We also modified Algorithm 6.1 to call Algorithm 5.1 to compute refined

Ritz vectors. All other parameters, in Algorithm 6.1 remained the same. Figure 2d displays the results and shows using refined Ritz vectors in place of iterative refined Ritz vectors performed poorly.

2.7 Numerical Examples

This section presents some numerical examples that illustrate the performance of Algorithm 6.1. For ease of comparisons we implemented Algorithm 6.1 in a MATLAB code called `trreigs`[1]. We compare our method to the publicly available MATLAB code `irbleigs`[1][1], the MATLAB interfaced code `primme_eigs`[23][2], and MATLAB's built–in function `eigs`. We refer the reader to the citations and noted websites for full details and descriptions of parameters. There are numerous selections and variety of combinations of parameters for each code. Some choices and combinations yield faster convergence than others. We cannot provide examples with all possible combinations. We used either the default values for the parameters or parameter choices that represent the fairest comparison with respect to Lanczos basis size and similarity with respect to the foundational Lanczos method for `trreigs`. All examples and methods used a common unit length vector with random entries that are normally distributed entries with zero mean.

The parameters for `trreigs` are based on Algorithm 6.1. For all examples we set $\delta_1 := \epsilon^{0.1}$, $\delta_2 := 0.9$, and maximum iterations 100 for Algorithm 5.1. The number of eigenpairs k, maximum size of the Lanczos basis m, tolerance for convergence ϵ, and location of eigenvalues are set depending on the example. In addition to checking the Ritz residual norm (8) for termination, the code `trreigs` also checks

$$\|AP_m\hat{q}_j - \hat{\rho}_j P_m\hat{q}_j\| \;=\; \sqrt{(T_m\hat{q}_j - \hat{\rho}_j\hat{q}_j)^T(T_m\hat{q}_j - \hat{\rho}_j\hat{q}_j) + \beta_m^2(e_m^T\hat{q}_j)^2} \leq \epsilon\|A\| \quad (68)$$

where $\hat{q}_j$ is the j^{th} column of the QR factorization of $[\hat{v}_1, \ldots, \hat{v}_k]$. We use the QR

[1]Code available at: `http://www.math.uri.edu/~jbaglama`
[2]Code available at: `https://github.com/primme/primme`

factorization to ensure that the eigenvectors are orthogonal. Furthermore, to help avoid the pitfalls of Example 5.2, we only check (68) when $|x_j^T \hat{z}_j| > \delta_2$. The technique of including additional vectors ($> k$) can greatly accelerate the convergence in restarted methods, like thick–restarting with Ritz vectors. There are many strategies for determining the number of restart vectors, see e.g. [25, 28]. A comparison of heuristic techniques is given in [28]. We implemented a simple but often effective strategy when the hybrid scheme uses thick–restarted, we restart with

$$k = \max(\text{floor}(nc + (m - nc)/2), k) \tag{69}$$

vectors where n_c is the number of converged desired approximate eigenvectors. However, using more than k vectors in the restarting scheme for iterative refined part was found to be counterproductive, often not satisfying the criteria in Algorithm 6.1 for switching. Since $m \ll n$ the code `trreigs` uses full reorthogonalization as outlined in step 8 of Algorithm 2.1. This is is a simple strategy, but can increase overall computational times.

The MATLAB code `irbleigs` is a block Lanczos method that uses the implicitly restarted formulas to apply Leja points as shifts. Given a Lanczos basis with m blocks, the method applies m Leja shifts via the implicit shift formulas until a single block of vectors is obtained and then restarts. Although the method utilizes implicit formulas for applying shifts to obtain a starting block, the overall structure can be considered an explicit restarted Lanczos method. For fair comparisons, `irbleigs` should be restricted to block size one, however that restriction often caused abnormally large number of matrix–vector products or no convergence. Therefore, in order to provide the fairest comparison with `irbleigs`, when appropriate, we recorded results with block size greater than one where the combination with the number of blocks is equal to the maximum size of the Lanczos basis m. Block size and number of blocks for `irbleigs` are reported in Table 3 as (block size, number of blocks). The common

starting vector is used, where the rest of the starting block is filled in with random vectors. Besides the number k of desired eigenpairs and tolerance for convergence we used the default settings for all other parameters.

MATLAB's built–in function `eigs` used symmetric parameter true and Lanczos basis max size as m, and all default settings except the number k of desired eigenpairs, convergence tolerance, and common starting vector.

The MATLAB interfaced code `primme_eigs` uses the state–of–the–art high performance C99 library PRIMME for computing the eigenvalues and eigenvectors. This is an impressive, carefully designed code that includes numerous parameter settings, multiple routines/techniques, and options to include preconditioning. There are 15 choices for methods. For comparisons, we used the setting for method to be "default_min_matvecs" (referred to as min_mv in Table 3) which is the best method for heavy matrix-vector products and performed better than the default "dynamic" for Examples 7.1 to 7.4. On some m choices for Example 7.5, "dynamic" performed better, therefore we reported the better results for Example 7.5 ("dynamic" is referred to as dyn in Table 3). We also included the method "jdqmr_etol" (referred to as jd_tol in Table 3). `primme_eigs` allows the user to input preconditioners to accelerate convergence. We did not apply any preconditioners for the reported examples. We set the parameters "isreal" and "isdouble" to be true and used k for desired eigenpairs and the common tolerance for convergence. Unless specified in the example, we used the default values for all other adjustable parameters. `primme_eigs` allows the user to include any number of initial guesses to the eigenvectors, however we only set a starting vector to the routine to be the common starting vector used for all routines in that example. `primme_eigs` allows the user to select the maximum size of the search subspace. We set the maximum size of the search subspace to be the common restrict value m for the other routines. It should be noted, the storage requirement

and maximum size of the search space for `primme_eigs` are not always equivalent, see [23, Section 3.4.1].

All examples use the location of eigenvalues to be largest in magnitude. Example 7.1 also includes an example for smallest algebraic. In all examples, the matrix A was only accessed by call to a function with input x and output Ax. In the Table 3, the cpu times are in seconds recorded using MATLAB's tic-toc command. The row for error represents $\max \|Ax_i - \lambda_i x_i\|$ which was computed outside the routines with the outputted approximations. The references to (62) and (63) refer to the different matrices used to find the coefficients for `trreigs`. Similar to Example 6.1, using (58) reported inferior results and is not recorded. We finally remark that the performance of the methods in our comparisons also depends on the machine architecture, MATLAB coding style, and numerical implementation, (e.g. selective, partial or full reorthgonalization). The MATLAB code `trreigs` is only an illustration of Algorithm 6.1 and was not designed in the same fashion as the publicly/commercially developed codes. Nevertheless, the examples do show that Algorithm 6.1 can match or outperform the performance of the other methods. All numerical examples were performed on matrices from SuiteSparse Matrix Collection [4] and all computations were carried out using MATLAB version R2019b on an iMac with 3.7Ghz Intel Core i5 processor and 32GB (2667 MHz) of memory using operating system macOS Mojave. Machine epsilon is $\epsilon = 2.2 \cdot 10^{-16}$.

Example 2.7.1. We considered two matrices, 2,680 $\times$ 2,680 *dwt2680* and 2,233 $\times$ 2,233 *lshp2233* that were used as numerical examples in [8]. For *dwt2680* the author was seeking 5 dominant eigenvalues and for *lshp2233* the 5 smallest algebraic eigenvalues. Both examples used a stopping criteria of 10^{-6}. The examples in [8] compared several related methods, the implicitly restarted Arnoldi (IRA), the implicitly restarted refined Arnoldi (IRRA), and the implicitly restarted refined har-

monic Arnoldi (IRRHA). As a point of reference the best computed result for *dwt2680* for the smallest used space $m = 20$, was for the IRRA with 244 mvp, [8, Table 7] and the best computed result for *lshp2233* for the smallest used space $m = 20$, was for the IRRHA with 1333 mvp, [8, Table 6]. Table 4a displays the results for *dwt2680* and Table 4b displays the results for *lshp2233*. For both matrices and all methods we used $k = 5$ and $\epsilon = 10^{-6}$. For *dwt2680* we display results for $m = 10, 20$ and for *lshp2233* for $m = 20$. The code `trreigs` displays the best results with respect to mvp for *lshp2233* and when $m = 10$ for *dwt2680* with comparable results when $m = 20$.

Example 2.7.2. We considered the 12,992 $\times$ 12,992 matrix *tuma2*. This was the only symmetric matrix that was used as a numerical example in [12] for finding the 6 dominant eigenvalues with a stopping criteria of 10^{-10}. The example in [12] compared several related methods, thick–restarted block Arnoldi, modified thick–restarted block Arnoldi, a hybrid modified Ritz thick–restarted and refined block Arnoldi method, and the block Krylov-Schur algorithm [30]. As a point of reference the best computed result with respect to mvp for the smallest used space $m = 18$, was for the modified thick–restarted block Arnoldi with 1520 mvp, [12, Table 6]. Table 4c displays the results for $k = 5$, $m = 10, 18$ and $\epsilon = 10^{-10}$. The code `trreigs` displays competitive results.

Example 2.7.3 We considered the 256,000 $\times$ 256,000 matrix *Lin* that was used in Example 6.1 in Section 2.6.1. We are searching for largest eigenvalue(s) and associated vector(s). We compared the codes with $k = 1, 4$, $m = 15$, and $\epsilon = 10^{-8}$. Table 4d displays the results. Notice that the results are significantly better than the recorded results in Fig. 1. This is due in part to using the strategy (69) for the thick–restarted scheme and incorporating stopping criteria (68). The code `trreigs` displays the best results with respect to mvp when compared to `eigs` and `irbleigs` for both $k = 1$ and $k = 4$.

Example 2.7.4 We considered the 1,062,400 × 1,062,400 matrix *nlpkkt80*. We are searching for largest eigenvalue and associated eigenvector while using the smallest search space. We set $\epsilon = 10^{-6}$. `eigs` did not record convergence within 6000 matrix-products until $m = 10$. For m equal to 3 for `primme_eigs` we set the parameters $maxPrevRetain = 1$ and $minRestartSize = 1$, otherwise they were set as the default values. The reported largest eigenvalue was 259.799. We include results for `trreigs` not requiring (67) restriction. This column is labeled 'FLT' under `trreigs`. Table 4e displays the results. The code `trreigs` with 'FLT' displays the best results with respect to mvp and smallest space $m = 2$.

Example 2.7.5 We considered the 214,005,017 × 214,005,017 matrix *kmer_V1r*. We are searching for largest eigenvalue and associated eigenvector while using the smallest possible search space. We set $\epsilon = 10^{-6}$ and used the smallest possible value for m for each routine to get convergence within 200 matrix–vector products. The Matlab code, `irbleigs` did not converge for $m = 3, 4, 5$ and jdqmr_etol did not converge for $m = 3, 4$ and therefore are not reported. For m equal to 3 for `primme_eigs` we set the parameters $maxPrevRetain = 1$ and $minRestartSize = 1$, otherwise they were set as the default values. We include results for `trreigs` not requiring (67) restriction. This column is labeled 'FLT' under `trreigs`. The reported largest eigenvalue was 6.50346. Table 4f displays the results. For the smallest space $m = 2$ the code `trreigs` with 'FLT' displays competitive results.

2.8 Conclusions

This paper presents a restarted hybrid method that combines thick–restarting with restarting with a linear combination of iterative refined Ritz vectors. The method does not require factorization of A, and can therefore be applied to very large problems. Numerical examples show the method to be competitive with other available codes with respect to matrix–vector products and storage required.

Appendix

Lemma 2.8.1. *Given* $\|y\| = 1$ *and* $\|x\| = 1$ *then* $\|x - (x^T y)y\| \leq 1$. *Additionally, if* $x^T y \neq 0$ *then* $\|x - (x^T y)y\| < 1$.

Proof. Follows from $\|x - (x^T y)y\|^2 = 1 - (x^T y)^2$ and $0 \leq (x^T y)^2 \leq \|x\|^2 \|y\|^2 = 1$. □

List of References

[1] J. Baglama, D. Calvetti, and L. Reichel, *Irbl: An implicitly restarted block-lanczos method for large-scale hermitian eigenproblems*, SIAM Journal on Scientific Computing, 24 (2003), pp. 1650–1677.

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the solution of algebraic eigenvalue problems: a practical guide*, SIAM, 2000.

[3] Y. Cai, L.-H. Zhang, Z. Bai, and R.-C. Li, *On an eigenvector-dependent nonlinear eigenvalue problem*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1360–1382.

[4] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), pp. 1–25.

[5] S. Feng and Z. Jia, *A refined jacobi-davidson method and its correction equation*, Computers & Mathematics with Applications, 49 (2005), pp. 417–427.

[6] Z. Jia, *Refined iterative algorithms based on arnoldi's process for large unsymmetric eigenproblems*, Linear algebra and its applications, 259 (1997), pp. 1–23.

[7] Z. Jia, *Polynomial characterizations of the approximate eigenvectors by the refined arnoldi method and an implicitly restarted refined arnoldi algorithm*, Linear algebra and its applications, 287 (1999), pp. 191–214.

[8] Z. Jia, *The refined harmonic arnoldi method and an implicitly restarted refined algorithm for computing interior eigenpairs of large matrices*, Applied numerical mathematics, 42 (2002), pp. 489–512.

[9] Z. Jia, *Some theoretical comparisons of refined ritz vectors and ritz vectors*, Science in China Series A: Mathematics, 47 (2004), pp. 222–233.

[10] Z. Jia, *The convergence of harmonic ritz values, harmonic ritz vectors and refined harmonic ritz vectors*, Mathematics of computation, 74 (2005), pp. 1441–1456.

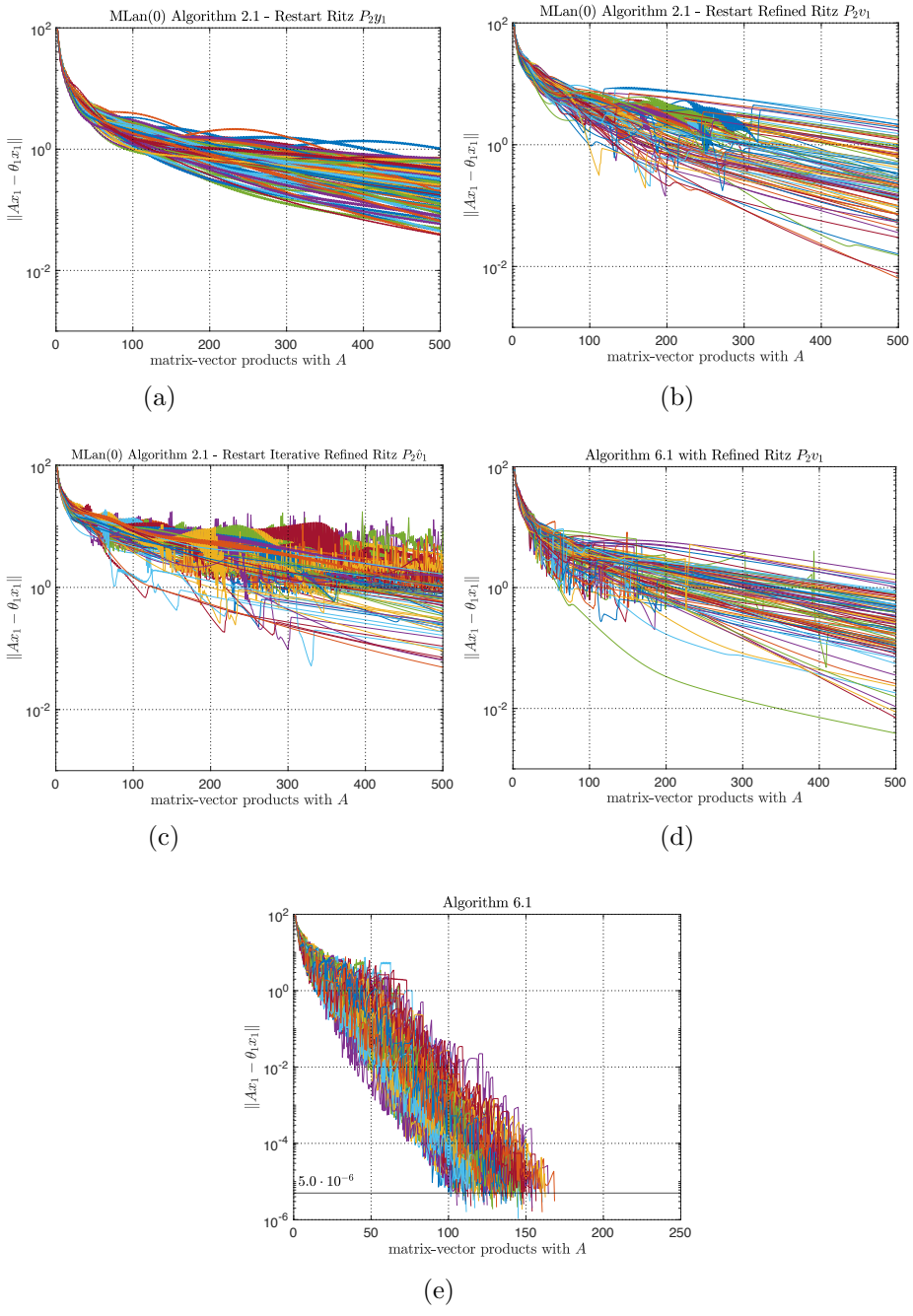


Figure 2. Example 2.5.3 and 2.6.2. Matrix $A = \text{diag}(1:500)$ is a 500×500 diagonal matrix. We are searching for the largest eigenpair and set $m = 2$. The figures (a), (b), and (c) use MLan(0) Algorithm 2.1 where the (re)starting vector p_1 on the next restart to be Ritz vector P_2y_1, refined Ritz P_2v_1, and iterative refined Ritz $P_2\hat{v}_1$, respectively. The figure (d) Algorithm 6.1 but with refined Ritz vectors in place of iterative refined Ritz vectors and figure (e) uses Algorithm 6.1 as presented. The example is done 100 times for each figure with a different beginning random vector. Each line represent a start with a random vector and then a restart using the stated vector. The process was terminated at 500 matrix-vector products or when $\|Ax_1 - \theta_1 x_1\| \leq 10^{-8}\|A\|$. Only Algorithm 6.1 as presented converged, figure (e).

[11] Z. Jia and G. W. Stewart, *On the convergence of ritz values, ritz vectors, and refined ritz vectors*, (1999).

[12] W. Jiang and G. Wu, *A thick-restarted block arnoldi algorithm with modified ritz vectors for large eigenproblems*, Computers & Mathematics with Applications, 60 (2010), pp. 873–889.

[13] R. B. Lehoucq and D. C. Sorensen, *Deflation techniques for an implicitly restarted arnoldi iteration*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 789–821.

[14] R. Li, Y. Xi, E. Vecharynski, C. Yang, and Y. Saad, *A thick-restart lanczos algorithm with polynomial filtering for hermitian eigenvalue problems*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2512–A2534.

[15] R. Morgan, *On restarting the arnoldi method for large nonsymmetric eigenvalue problems*, Mathematics of Computation, 65 (1996), pp. 1213–1230.

[16] R. B. Morgan, *Implicitly restarted gmres and arnoldi methods for nonsymmetric systems of equations*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1112–1135.

[17] B. N. Parlett, *The symmetric eigenvalue problem*, vol. 20, siam, 1998.

[18] M. Ravibabu and A. Singh, *On refined ritz vectors and polynomial characterization*, Computers & Mathematics with Applications, 67 (2014), pp. 1057–1064.

[19] Y. Saad, *Variations on arnoldi's method for computing eigenelements of large unsymmetric matrices*, Linear algebra and its applications, 34 (1980), pp. 269–295.

[20] H. D. Simon, *Analysis of the symmetric lanczos algorithm with reorthogonalization methods*, Linear algebra and its applications, 61 (1984), pp. 101–131.

[21] G. L. Sleijpen and H. A. Van der Vorst, *A jacobi–davidson iteration method for linear eigenvalue problems*, SIAM review, 42 (2000), pp. 267–293.

[22] D. C. Sorensen, *Implicit application of polynomial filters in a k-step arnoldi method*, Siam journal on matrix analysis and applications, 13 (1992), pp. 357–385.

[23] A. Stathopoulos and J. R. McCombs, *Primme: preconditioned iterative multimethod eigensolver methods and software description*, ACM Transactions on Mathematical Software (TOMS), 37 (2010), p. 21.

[24] A. Stathopoulos and Y. Saad, *Restarting techniques for the (jacobi-) davidson symmetric eigenvalue methods*, Electron. Trans. Numer. Anal, 7 (1998), pp. 163–181.

[25] A. Stathopoulos, Y. Saad, and K. Wu, *Dynamic thick restarting of the davidson, and the implicitly restarted arnoldi methods*, SIAM Journal on Scientific Computing, 19 (1998), pp. 227–245.

[26] G. W. Stewart, *A krylov–schur algorithm for large eigenproblems*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), pp. 601–614.

[27] K. Wu and H. Simon, *Thick-restart lanczos method for large symmetric eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications, 22 (2000), pp. 602–616.

[28] K. Wu and H. D. Simon, *Dynamic restarting schemes for eigenvalue problems*, tech. report, Lawrence Berkeley National Lab., CA (US), 1999.

[29] L. Wu, F. Xue, and A. Stathopoulos, *Trpl+ k: Thick-restart preconditioned lanczos+ k method for large symmetric eigenvalue problems*, SIAM Journal on Scientific Computing, 41 (2019), pp. A1013–A1040.

[30] Y. Zhou and Y. Saad, *Block krylov–schur method for large symmetric eigenvalue problems*, Numerical Algorithms, 47 (2008), pp. 341–359.

Table 3. Numerical Examples

	trreigs				irbleigs		eigs		primme_eigs	
m	10		20		10	20	10	20	10	
	(62)	(63)	(62)	(63)	(1,10)	(1,20)			min_mv	jd_tol
mvp	98	106	94	129	140	188	104	86	102	173
cpu	0.10s	0.06s	0.04s	0.05s	0.16s	0.06s	0.05s	0.02s	0.04s	0.02s
err	$7.9\cdot10^{-6}$	$7.6\cdot10^{-6}$	$6.1\cdot10^{-6}$	$9.4\cdot10^{-7}$	$8.0\cdot10^{-6}$	$7.7\cdot10^{-6}$	$6.9\cdot10^{-6}$	$2.3\cdot10^{-6}$	$1.1\cdot10^{-5}$	$1.1\cdot10^{-5}$

(a) Example 2.7.1. 2,680 × 2,680 matrix *dwt2680* ($k = 5$). Largest in magnitude.

	trreigs		irbleigs	eigs	primme_eigs	
	(62)	(63)	(4,5)		min_mv	jd_tol
mvp	497	435	920	510	456	687
cpu	0.34s	0.27s	0.23s	0.08s	0.08s	0.05s
err	$2.8\cdot10^{-6}$	$2.5\cdot10^{-6}$	$3.2\cdot10^{-6}$	$8.8\cdot10^{-7}$	$6.8\cdot10^{-6}$	$4.1\cdot10^{-6}$

(b) Example 2.7.1. 2,233 × 2,233 matrix *lshp2233* ($k = 5$, $m = 20$). Smallest algebraic.

	trreigs				irbleigs		eigs		primme_eigs	
m	10		18		10	18	10	18	10	
	(62)	(63)	(62)	(63)	(2,5)	(3,6)			min_mv	jd_tol
mvp	647	498	264	264	488	561	713	240	408	586
cpu	0.56s	0.44s	0.26s	0.25s	0.34s	0.25s	0.29s	0.09s	0.21s	0.12
err	$2.6\cdot10^{-10}$	$2.2\cdot10^{-10}$	$4.3\cdot10^{-10}$	$4.3\cdot10^{-10}$	$1.9\cdot10^{-7}$	$1.2\cdot10^{-8}$	$4.4\cdot10^{-10}$	$4.8\cdot10^{-10}$	$4.5\cdot10^{-10}$	$3.9\cdot10^{-10}$

(c) Example 2.7.2. 12,992 × 12,992 matrix *tuma2* ($k = 6$). Largest in magnitude.

	trreigs			irbleigs		eigs		primme_eigs			
k	1	4		1	4	1	4	1		4	
		(62)	(63)	(3,5)	(3,5)			min_mv	jd_tol	min_mv	jd_tol
mvp	491	970	1129	1170	1953	839	1095	380	449	649	874
cpu	6.89s	13.85s	18.56s	8.73s	16.73s	4.05s	4.64s	3.62s	1.69s	6.70s	3.22s
err	$7.6\cdot10^{-6}$	$9.5\cdot10^{-6}$	$7.8\cdot10^{-6}$	$7.9\cdot10^{-6}$	$1.0\cdot10^{-5}$	$8.3\cdot10^{-6}$	$5.0\cdot10^{-6}$	$9.7\cdot10^{-6}$	$1.1\cdot10^{-5}$	$1.1\cdot10^{-5}$	$1.1\cdot10^{-5}$

(d) Example 2.7.3. 256,000 × 256,000 matrix *Lin* ($m = 15$). Largest in magnitude.

	trreigs		irbleigs	eigs	primme_eigs			
m	2	2	3	10	3		4	
		FLT	(1,3)		min_mv	jd_tol	min_mv	jd_tol
mvp	1098	692	1422	5800	2014	1507	742	1010
cpu	48.73s	30.62s	58.08s	227.66s	81.45s	49.22s	31.94s	32.97s
err	$2.4\cdot10^{-4}$	$2.6\cdot10^{-4}$	$1.8\cdot10^{-4}$	$2.6\cdot10^{-4}$	$2.5\cdot10^{-4}$	$2.2\cdot10^{-4}$	$2.6\cdot10^{-4}$	$2.3\cdot10^{-4}$

(e) Example 2.7.4. 1,062,400 × 1,062,400 matrix *nlpkkt80* ($k = 1$). Largest in magnitude.

	trreigs		eigs			primme_eigs			
m	2	2	3	4	5	3	4	5	
		FLT				min_mv	dyn	dyn	jd_tol
mvp	98	70	131	132	98	60	58	69	80
cpu	987.81s	650.78s	599.94s	654.22s	490.73s	370.15s	493.21s	2998.2s	2225.7s
err	$6.4\cdot10^{-6}$	$5.4\cdot10^{-6}$	$6.3\cdot10^{-6}$	$5.5\cdot10^{-6}$	$5.5\cdot10^{-6}$	$5.2\cdot10^{-6}$	$4.6\cdot10^{-6}$	$4.8\cdot10^{-6}$	$3.1\cdot10^{-6}$

(f) Example 2.7.5. 214,005,017 × 214,005,017 matrix *kmer_V1r* ($k = 1$). Largest in magnitude.

CHAPTER 3

Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods

James Baglama, Vasilije Perović, and Jennifer Picucci
Department of Mathematics and Applied Mathematical Sciences
University of Rhode Island, Kingston, Rhode Island 02881-0816, USA

This manuscript was submitted for publication to Numerical Algorithms on July 30, 2021.

Keywords: Singular value computation, partial singular value decomposition, iterative method, large-scale computation, Krylov subspace, Refined Ritz
Mathematics Subject Classification (2010): 65F15, 65F50, 15A18

Abstract This paper describes hybrid restarted Lanczos bidiagonalization methods for the computation of a few of the largest (or smallest) singular triplets of very large scale matrices. Restarting is carried out either by a thick–restarted scheme with Ritz or harmonic Ritz vectors or explicitly restarted with iterative refined Ritz vectors. Several criteria are used to determine which restarted scheme is to be used. The iterative refined Ritz vectors are computed using a scheme in which the refined process is repeated until convergence. In the context of the symmetric eigenvalue computation, the authors have shown that this iterative scheme converges and moreover it is highly effective when combined with thick–restarting in the cases when memory is limited. Given the connections with the Lanczos tridiagonal process, results are carried over in this context of computing singular triplets. Also presented, are MATLAB codes that implement the described algorithms along with numerous examples demonstrating our methods are competitive with other available routines.

3.1 Introduction

The *singular value decomposition* (SVD) of matrix $A \in \mathbb{R}^{\ell \times n}$ $(\ell \geq n)$[1] is a factorization of the form

$$A = U\Sigma V^T \tag{70}$$

where $U = [u_1, \ldots, u_n] \in \mathbb{R}^{\ell \times n}$ and $V = [v_1, \ldots, v_n] \in \mathbb{R}^{n \times n}$ have orthonormal columns and $\Sigma = \operatorname{diag}\big(\sigma_1, \sigma_2, \ldots, \sigma_n\big) \in \mathbb{R}^{n \times n}$ such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0\,.$$

The σ_j's are referred to as the singular values of A, while u_j's and v_j's are the corresponding left and right singular vectors of A, respectively. Collectively, $\{\sigma_j, u_j, v_j\}$ is referred to as a *singular triplet* of A. From (70), for $0 < s \leq n$, we have

$$AV_s = U_s\Sigma_s, \qquad A^TU_s = V_s\Sigma_s, \tag{71}$$

where $\Sigma_s = \operatorname{diag}\big(\sigma_1, \sigma_2, \ldots, \sigma_s\big) \in \mathbb{R}^{s \times s}$, $U_s = [u_1, \ldots, u_s] \in \mathbb{R}^{\ell \times s}$, and $V_s = [v_1, \ldots, v_s] \in \mathbb{R}^{n \times s}$; when $s < n$ we refer to the factorization (71) as a *partial singular value decomposition* of A, or s-PSVD for short.

The primary focus of this paper is on computing a small number of singular triplets, let's say k, corresponding to the largest singular values and associated vectors, while using as little memory as possible. In other words, we are interested in computing $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$ such that

$$Av_j = \sigma_j u_j, \qquad A^Tu_j = \sigma_j v_j, \qquad j = 1, 2, \ldots, k\,, \tag{72}$$

or equivalently, computing a k-PSVD of A.

Despite the fact that the SVD of a matrix can be traced back to the 1870's [47], it took nearly a century for it to become one of the most-widely used matrix factorizations. This is largely due to the seminal work by Golub and Kahan [13]

[1] Otherwise replace A with A^T.

creating the Golub-Kahan-Lanczos (GKL) bidiagonalization procedure where they showed that singular triplets can be computed efficiently and in a numerically stable way. Today, SVD is one of the main computational methods with numerous applications, e.g., dimension reduction, Principal Component Analysis (PCA), machine learning [28, 49, 50], genomics [1, 4, 43], data mining, data visualization, and detection of patterns [11, 18, 38]. Many of the matrices arising from these applications are typically very large, sparse and only accessible via matrix-vector routines which makes it impractical for the computation of the entire singular structure. Fortunately, in many cases one is only interested in a few largest (or smallest) singular triplets. The computation of only a few of the extreme singular triplets of very large sparse matrices has spurred a considerable amount of research and software, see e.g., [5, 6, 9, 12, 15, 24, 25, 29, 30, 31, 32, 33, 53] and the references therein.

One of the features shared by many of the referenced routines is the vital role played by the GKL procedure [13]. Recall that for some starting unit vector p_1 (and $q_1 := Ap_1$), this procedure creates orthonormal bases for the Krylov subspaces,

$$\begin{aligned} \mathbb{K}_m(A^TA, p_1) &= \text{span}\left\{p_1,\, A^TAp_1,\, \left(A^TA\right)^2 p_1,\, \ldots,\, \left(A^TA\right)^{m-1} p_1\right\}, \\ \mathbb{K}_m(AA^T, q_1) &= \text{span}\left\{q_1,\, AA^Tq_1,\, \left(AA^T\right)^2 q_1,\, \ldots,\, \left(AA^T\right)^{m-1} q_1\right\}, \end{aligned} \tag{73}$$

using only matrix-vector products with A and A^T while avoiding explicitly creating the matrices A^TA and AA^T. This makes the process ideal for very large scale problems. The GKL procedure at step m yields the m-GKL factorization,

$$AP_m = Q_mB_m\,, \tag{74}$$

$$A^TQ_m = P_mB_m^T + fe_m^T = \left[\, P_m \;\; p_{m+1} \,\right] \left[\begin{array}{c} B_m^T \\ \beta_m e_m^T \end{array}\right], \tag{75}$$

where the matrices $P_m = [p_1, \ldots, p_m] \in \mathbb{R}^{n\times m}$ and $Q_m = [q_1, \ldots, q_m] \in \mathbb{R}^{\ell\times m}$ have orthonormal columns which form bases for Krylov subspaces (73) respectively, the residual vector $f \in \mathbb{R}^n$ satisfies $P_m^Tf = 0$, $\beta_m = \|f\|$, and $p_{m+1} = f/\beta_m$. Further, e_m

is the m^{th} axis vector of appropriate dimension and,

$$B_m := \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \alpha_3 & \beta_3 & & \\ & & & \ddots & \ddots & \\ & & & & \ddots & \beta_{m-1} \\ & & & & & \alpha_m \end{bmatrix} \in \mathbb{R}^{m \times m} \tag{76}$$

is an upper bidiagonal matrix. Now approximations of the singular triplets of A can be obtained from the singular triplets of B_m. Observe that when the norm of the residual vector f is small, the singular values of B_m are close to the singular values of A (exact when f vanishes) and the associated singular vectors are computed using the basis vectors of the Krylov subspaces, see Section 3.2 for details. However, these approximations are typically poor for modest values of m, hence either requiring m to be increased or the starting vector p_1 to be modified (explicitly or implicitly) and the GKL process restarted. Considering that the matrix A is of large scale and assuming prohibitive memory limitations, increasing m to a suitable value to get acceptable approximations is not an option. Therefore, much of the research, including this paper, revolves around developing different restarting schemes for the GKL process. Note though that there are already several notable routines that do this [5, 6, 24, 25, 29, 30], particularly the thick–restarted GKL routine in [5] which plays a vital role in this paper.

In [5], Baglama and Reichel exploited the mathematical equivalence for symmetric eigenvalue computations of the implicitly restarted Arnoldi (Lanczos) method of Sorensen [45] and the thick–restarting scheme of Wu and Simon [52] and applied it to a restarted GKL procedure; for details on the equivalences for eigenvalue computations and in the context of least squares see [34] and [7], respectively. Their thick–restarted GKL routine turns out to be a simple and computationally fast method for computing a few of the extreme singular triplets of large matrices that is less sensitive to propagated round-off errors; for a brief review of this scheme see Section 3.2. However,

the routine struggles when the dimension, m, of the Krylov subspaces is memory limited and kept relatively small in relationship to the number of desired singular triplets k, see the examples in Section 3.5. Recently, in the context of symmetric eigenvalue computation, the authors overcame this memory restriction by creating a hybrid restarted Lanczos method that combines thick–restarting with Ritz vectors with a new technique, iteratively refined Ritz vectors [2]. The thick–restarted part was carried out as described in [52] and when certain criteria were met, the routine switched to restarting with a linear combination of iteratively refined Ritz vectors. In [2], the authors showed that the scheme of thick–restarting of Wu and Simon was not available with refined or iteratively refined Ritz vectors. An alternate restarting scheme when using iteratively refined Ritz vectors was derived based on the relationships outlined by Morgan in [34]. Morgan showed in the case of Ritz values/vectors that implicitly restarting was equivalent to restarting with a certain linear combination of Ritz vectors. Therefore, in a similar way, we chose constants to linearly combine the iteratively refined Ritz vectors to restart the process. The idea is that this linear combination of the iteratively refined Ritz vectors will resemble a restart, in a somewhat asymptotic sense, of thick–restarting, see [2, Sec. 6] for details.

It is well–known that the refined Ritz vectors can provide better eigenvector approximations than the Ritz vectors, see analysis [22, 26] for details. But in a restarted scheme, "better" approximation is only a part of the overall need and an efficient restarting scheme is also required. One approach was presented in [20], where Jia used "refined" shifts in the implicitly restarted Arnoldi method. In the context of SVD, this approach was extended by Jia and Niu resulting in an implicitly restarted GKL procedure for computing singular triplets [24, 25]. In this paper, we present another approach where we extend the restarted hybrid iterative refined scheme from [2] to the GKL procedure for computing singular triplets.

In the context of the symmetric eigenvalue problem, the authors in [2] consider an iterative refined Ritz scheme in which the refined process is repeated until convergence. This process has the benefit of eliminating part of the refined Ritz residuals and aiding in the ability to create a linear combination to resemble thick–restarting, all while producing a "smaller" norm. A brief review of the iterative refined Ritz scheme is provided in Section 3.3 though for a thorough discussion and results we refer the reader to [2].

To make the connection between the symmetric eigenvalue problem and the SVD of $A \in \mathbb{R}^{\ell \times n}$ more explicit, we consider the matrices

$$A^T A \in \mathbb{R}^{n \times n} \qquad \text{and} \qquad C \,=\, \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \in \mathbb{R}^{(\ell+n)\times(\ell+n)}. \tag{77}$$

We refer to $A^T A$ as the normal matrix or system and C as the augmented matrix or system. The eigenvalues of the normal matrix $A^T A$ are the squares of singular values of A, while the associated eigenvectors of $A^T A$ are the corresponding right singular vectors of A, i.e., $A^T A v_j = \sigma_j^2 v_j$. When $\sigma_j \neq 0$, the left singular vectors can be computed as $u_j = (1/\sigma_j)\, A v_j$. We note that from a practical standpoint, good SVD algorithms based on the connection with $A^T A$ typically work on A directly due to the fact that the explicit construction of $A^T A$ (or $A A^T$) is numerically unstable. In the case of the augmented matrix C, its eigenvalues are $\pm\sigma_j$ as well as $\ell - n$ zero eigenvalues. The eigenvectors of C associated with $\pm\sigma_j$ are $\frac{1}{\sqrt{2}}[u_j^T \,,\, \pm v_j^T]^T$, where $\{\sigma_j, u_j, v_j\}$ is a singular triplet of A.

Now observe that by multiplying equation (74) from the left by A^T produces the Lanczos tridiagonal decomposition of the normal matrix $A^T A$, namely

$$A^T A P_m \,=\, P_m B_m^T B_m + \alpha_m f_m e_m^T \,=\, \begin{bmatrix} P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix}. \tag{78}$$

Similarly, in the case of the augmented matrix C, after performing $2m$ steps of the standard Lanczos algorithm with the starting vector $[0^T \,,\, p_1^T]^T \in \mathbb{R}^{\ell+n}$ one obtains a

$2m \times 2m$ tridiagonal projection matrix, which when followed by an odd-even permutation gives the following Lanczos factorization [14, Sec. 10.4.3] [29]

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m & 0 \\ 0 & P_m \end{bmatrix} = \begin{bmatrix} Q_m & 0 \\ 0 & P_m \end{bmatrix} \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ f_m e_m^T & 0 \end{bmatrix}, \quad (79)$$

$$= \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \\ \beta_m e_m^T & 0 \end{bmatrix}.$$

Finally, considering the Lanczos factorization relationships (78) and (79), the results and properties related to the hybrid iterative refined Ritz scheme in [2] are carried over to the methods developed in the subsequent sections. Although our development is focused on the largest singular values, it can be applied to computing the smallest singular values and associated vectors, see Example 3.5.4. However, it should be noted that there are other routines with a specific focus on computing the smallest singular values, that utilize, for example preconditioners, factorization, or specific techniques, see e.g., [25, 29, 33, 53].

The paper is organized as follows. The thick–restarted scheme with Ritz or harmonic Ritz vectors is given in Section 3.2 while a review of iteratively refined Ritz vectors computed either on the normal system (78) or the augmented system (79) can be found in Section 3.3. In Section 3.4, we describe our new hybrid methods and present two algorithms for computing singular triplets. Numerical examples are presented in Section 3.5 followed by conclusions in Section 3.6.

Throughout this paper $\|\cdot\|$ denotes the Euclidean vector norm or the associated induced matrix norm. I_k is used to denote the $k \times k$ identity matrix while I_{k_1,k_2}, with $k_1 \geq k_2$, denotes the first k_2 columns of I_{k_1}; when the size is clear from the context we simply write I. When useful and for ease of presentation we utilize MATLAB's syntax (colon and semicolon) for constructing block matrices. An expression of the form $\xi := \eta$ (resp., $\xi =: \eta$) is used to denote that ξ *is defined to be equal to* η (resp., η *is defined to be equal to* ξ). In order to distinguish among numerous SVD computations and to

help the reader, throughout the paper we adopt the *convention* that superscripts (rz), (hm), $(rf\text{-}\star)$, and $(it\text{-}\star)$ correspond to the computations involving Ritz, harmonic Ritz, refined Ritz, and iteratively refined Ritz values/vectors, respectively; here $\star \in \{n, a\}$ denotes that (iteratively) refined Ritz are computed with respect to either the normal or the augmented systems (77). Finally, when a formula is developed and used in different settings, we use a "generic" superscript (..) (see Section 3.2.3).

3.2 Thick–restarted GKL process with Ritz or harmonic Ritz vectors

We briefly describe the method of thick–restarting with Ritz or harmonic Ritz vectors and refer the reader to [5] for a thorough discussion and details.

3.2.1 Thick–restarting with Ritz vectors

Let the s-PSVD of B_m from (76) be given as

$$B_m V_s^{(\mathrm{rz})} \;=\; U_s^{(\mathrm{rz})}\Sigma_s^{(\mathrm{rz})}\,, \qquad B_m^T U_s^{(\mathrm{rz})} \;=\; V_s^{(\mathrm{rz})}\Sigma_s^{(\mathrm{rz})}\,, \tag{80}$$

where $U_s^{(\mathrm{rz})} = [u_1^{(\mathrm{rz})}, \ldots, u_s^{(\mathrm{rz})}] \in \mathbb{R}^{m\times s}$ and $V_s^{(\mathrm{rz})} = [v_1^{(\mathrm{rz})}, \ldots, v_s^{(\mathrm{rz})}] \in \mathbb{R}^{m\times s}$ have orthonormal columns and $\Sigma_s^{(\mathrm{rz})} = \mathrm{diag}\left(\sigma_1^{(\mathrm{rz})}, \ldots, \sigma_s^{(\mathrm{rz})}\right) \in \mathbb{R}^{s\times s}$ such that $\sigma_1^{(\mathrm{rz})} \geq \sigma_2^{(\mathrm{rz})} \geq \cdots \geq \sigma_s^{(\mathrm{rz})} \geq 0$. Define $\tilde{P}_s := P_m V_s^{(\mathrm{rz})}$ and $\tilde{Q}_s := Q_m U_s^{(\mathrm{rz})}$, where P_m and Q_m are as in (74) and (75). Then from equations (74), (75), and (80) it follows that

$$\begin{aligned} A\tilde{P}_s = AP_mV_s^{(\mathrm{rz})} \;&=\; Q_mB_mV_s^{(\mathrm{rz})} \;=\; Q_mU_s^{(\mathrm{rz})}\Sigma_s^{(\mathrm{rz})}\,, \\ &=\; \tilde{Q}_s\Sigma_s^{(\mathrm{rz})} \;=:\; \tilde{Q}_s\tilde{B}_s\,. \end{aligned} \tag{81}$$

Similarly,

$$\begin{aligned} A^T\tilde{Q}_s = A^TQ_mU_s^{(\mathrm{rz})} \;&=\; P_mB_m^TU_s^{(\mathrm{rz})} + fe_m^TU_s^{(\mathrm{rz})} = P_mV_s^{(\mathrm{rz})}\Sigma_s^{(\mathrm{rz})} + f(e_m^TU_s^{(\mathrm{rz})})\,, \\ &=\; \left[\tilde{P}_s\ p_{s+1}\right]\begin{bmatrix} \Sigma_s^{(\mathrm{rz})} \\ \hline \rho_1\ \ \cdots\ \ \rho_s \end{bmatrix} =: \left[\tilde{P}_s\ p_{s+1}\right]\tilde{B}_{s,s+1}^T\,, \end{aligned} \tag{82}$$

where $p_{s+1} = f/\|f\|$ and $\rho_j = \|f\|U_s^{(\mathrm{rz})}(m, j)$. Note that the pair of factorizations (81)-(82) can be extended via Algorithm 6.2 with p_{s+1} as the starting vector to obtain

a new factorization similar to the m-GKL factorization (74)-(75); the noted difference is in the structure of B_m which is given by

$$B_m = \begin{bmatrix} \begin{bmatrix} \tilde{B}_{s,s+1} \end{bmatrix} & & & 0 \\ & \alpha_{s+1} & \beta_{s+1} & \\ & & \ddots & \ddots \\ & & & \ddots & \beta_{m-1} \\ 0 & & & & \alpha_m \end{bmatrix} \in \mathbb{R}^{m \times m}. \tag{83}$$

One can then continue the overall process of computing approximate singular triplets of A, by computing the s-PSVD of B_m in (83), setting up equations (81)-(82) and extending them via Algorithm 6.2 to an analog of (74)-(75); Algorithm 6.3 outlines this process.

Remark. *The pairs of factorizations* (74)-(75) *(with* B_m *as in* (76) *or* (83)*) and (81)-(82) play a central role in this paper. As such, throughout the rest of this paper, we refer to (81)-(82) and* (74)-(75) *as an* s-GKL *and an* m-GKL *factorizations, respectively. Note that due to the structure of matrices* $\tilde{B}_s$ (81) *and* $\tilde{B}_{s,s+1}$ (82)*, the pair (81)-(82) is not a GKL factorization in the classical sense, though it can be transformed into one [48].*

3.2.2 Thick–restarting with harmonic Ritz vectors

Thick-restarting for the GKL process can also be performed with harmonic Ritz vectors [5]. Over the last several decades, the topics of harmonic Ritz values/vectors and their applications have been extensively studied, see e.g., [3, 5, 6, 7, 8, 15, 16, 17, 21, 23, 25, 29, 34, 35, 36, 39] and the references therein. Harmonic Ritz values/vectors are typically used when the focus is on approximating the smallest (or interior) singular triplets. In our numerical examples with the hybrid scheme and limited memory restrictions (see Section 3.5), we observed that the harmonic Ritz also gave very good results when searching for the largest singular triplet often performing better than thick-restarting with Ritz vectors. This may not be as surprising when one considers

the connection between thick–restarting and implicitly shifting – see [7, 35] for discussions on the equivalency of shifting by unwanted harmonic Ritz values (smallest when searching for largest) is the same as augmenting with wanted harmonic Ritz vectors.

In the rest of this section, we provide a brief outline on thick–restarting with harmonic Ritz vectors and refer the reader to [5] for additional details. Our development utilizes the connection of GKL factorization with the normal system A^TA.

Starting with the factorization (78) and the assumption that B_m is nonsingular, we consider the symmetric matrix $B_mB_m^T + \beta_m^2 e_m e_m^T$. If (θ_j, g_j) is an eigenpair of $B_mB_m^T + \beta_m^2 e_m e_m^T$, that is,

$$\big(B_mB_m^T + \beta_m^2 e_m e_m^T\big) g_j \;=\; \theta_j g_j \,, \tag{84}$$

then $(\theta_j, P_mB_m^{-1}g_j)$ defines a *harmonic Ritz pair* of A^TA. It turns out that one can avoid having to explicitly construct $B_mB_m^T$ in order to compute harmonic Ritz pairs of A^TA by considering the related matrix $B_{m,m+1}$ given by

$$B_{m,m+1} \;:=\; \left[\; B_m \;\; \beta_m e_m \;\right] \in \mathbb{R}^{m\times(m+1)}. \tag{85}$$

Then for any $s < m$, the s-PSVD of $B_{m,m+1}$ is given by

$$B_{m,m+1}\hat{V}_s \;=\; U_s^{(\mathrm{hm})}\Sigma_s^{(\mathrm{hm})}\,, \qquad B_{m,m+1}^T U_s^{(\mathrm{hm})} \;=\; \hat{V}_s\Sigma_s^{(\mathrm{hm})}\,, \tag{86}$$

where $U_s^{(\mathrm{hm})} = [u_1^{(\mathrm{hm})}, \ldots, u_s^{(\mathrm{hm})}] \in \mathbb{R}^{m\times s}$ and $\hat{V}_s = [\hat{v}_1, \ldots, \hat{v}_s] \in \mathbb{R}^{(m+1)\times s}$ have orthonormal columns and $\Sigma_s^{(\mathrm{hm})} = \mathrm{diag}\left(\sigma_1^{(\mathrm{hm})}, \ldots, \sigma_s^{(\mathrm{hm})}\right) \in \mathbb{R}^{s\times s}$ such that $\sigma_1^{(\mathrm{hm})} \geq \sigma_2^{(\mathrm{hm})} \geq \cdots \geq \sigma_s^{(\mathrm{hm})} > 0$.

In order to set up the thick–restarted routine with s harmonic Ritz vectors one can exploit the relationship of the s-PSVD of $B_{m,m+1}$ in (86) and the following standard symmetric eigenvalue problem

$$B_{m,m+1}B_{m,m+1}^T U_s^{(\mathrm{hm})} = (B_mB_m^T + \beta_m^2 e_m e_m^T)U_s^{(\mathrm{hm})} = U_s^{(\mathrm{hm})}(\Sigma_s^{(\mathrm{hm})})^2, \tag{87}$$

where the diagonal entries of $(\Sigma_s^{(\mathrm{hm})})^2$ are the s harmonic Ritz values θ_j in (84). Then it follows directly from the properties of block matrix multiplication and definition $V_s^{(\mathrm{hm})} := B_m^{-1} U_s^{(\mathrm{hm})}$ that

$$P_m V_s^{(\mathrm{hm})} \;=\; P_m B_m^{-1} U_s^{(\mathrm{hm})} \;=\; P_m \left[I \;\; \beta_m B_m^{-1} e_m \right] \hat{V}_s \, (\Sigma_s^{(\mathrm{hm})})^{-1} \,, \tag{88}$$

where

$$\hat{V}_s \;=\; \begin{bmatrix} B_m^{-1} U_s^{(\mathrm{hm})} \Sigma_s^{(\mathrm{hm})} & -\beta_m B_m^{-1} e_m \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_s \\ e_{m+1}^T \hat{V}_s \end{bmatrix} . \tag{89}$$

Note that the columns $P_m v_j^{(\mathrm{hm})}$ of $P_m V_s^{(\mathrm{hm})}$, for $j = 1, \ldots, s$, are the harmonic Ritz vectors of $A^T A$ associated with θ_j in (84). Using (78), the definition of $V_s^{(\mathrm{hm})}$, and (87), the residual vector for the harmonic Ritz pair $(\theta_j, P_m v_j^{(\mathrm{hm})})$ is given as,

$$\begin{aligned} r_j = A^T A P_m v_j^{(\mathrm{hm})} - \theta_j P_m v_j^{(\mathrm{hm})} \;&=\; e_m^T B_m v_j^{(\mathrm{hm})} (f - \beta_m^2 P_m B_m^{-1} e_m) \,, \\ &=\; e_m^T B_m v_j^{(\mathrm{hm})} \beta_m (p_{m+1} - \beta_m P_m B_m^{-1} e_m) \,, \\ &=\; e_m^T u_j^{(\mathrm{hm})} \beta_m \left[P_m \;\; p_{m+1} \right] \begin{bmatrix} -\beta_m B_m^{-1} e_m \\ 1 \end{bmatrix} . \end{aligned} \tag{90}$$

From (90) we see that all of the residual vectors r_j are just multiplies of the same vector $r^{(\mathrm{hm})}$, where

$$r^{(\mathrm{hm})} \;:=\; p_{m+1} \;-\; \beta_m P_m B_m^{-1} e_m. \tag{91}$$

With the aim of creating formulas analogous to (81) and (82) we notice from (88) and (91) that

$$[P_m V_s^{(\mathrm{hm})} \Sigma_s^{(\mathrm{hm})} \;\; r^{(\mathrm{hm})}] \;=\; [P_m \; p_{m+1}] \begin{bmatrix} B_m^{-1} U_s^{(\mathrm{hm})} \Sigma_s^{(\mathrm{hm})} & -\beta_m B_m^{-1} e_m \\ 0 & 1 \end{bmatrix} . \tag{92}$$

In order to have the columns in (92) be orthonormal, we compute the QR-decomposition of the far right matrix in (92) as

$$\begin{bmatrix} B_m^{-1} U_s^{(\mathrm{hm})} \Sigma_s^{(\mathrm{hm})} & -\beta_m B_m^{-1} e_m \\ 0 & 1 \end{bmatrix} = \left[\begin{array}{c|c} \hat{V}_s^{(\mathrm{hm})} & \hat{v} \\ 0 & \end{array} \right] \hat{R}_{s+1} \,, \tag{93}$$

where $\hat{V}_s^{(\mathrm{hm})} \in \mathbb{R}^{m \times s}$, $(\hat{V}_s^{(\mathrm{hm})})^T \hat{V}_s^{(\mathrm{hm})} = I_s$, $\hat{v} \in \mathbb{R}^{m+1}$, $\hat{v}(1\!:\!m)^T \hat{V}_s^{(\mathrm{hm})} = 0$, $\hat{R}_{s+1} \in \mathbb{R}^{(s+1)\times(s+1)}$ is upper triangular, and define $\hat{R}_s := \hat{R}_{s+1}(1\!:\!s, 1\!:\!s)$. Similarly to our

discussion on thick–restarted with Ritz vectors, we redefine $\tilde{P}_s := P_m \hat{V}_s^{(\mathrm{hm})}$ and $\tilde{Q}_s := Q_m U_s^{(\mathrm{hm})}$, which together with (74) and (93) give us the following relation analogous to (81)

$$\begin{aligned} A\tilde{P}_s = AP_m\hat{V}_s^{(\mathrm{hm})} &= Q_m B_m \hat{V}_s^{(\mathrm{hm})} = Q_m U_s^{(\mathrm{hm})} \Sigma_s^{(\mathrm{hm})} \hat{R}_s^{-1} \\ &= \tilde{Q}_s \Sigma_s^{(\mathrm{hm})} \hat{R}_s^{-1} =: \tilde{Q}_s \tilde{B}_s. \end{aligned} \tag{94}$$

Now in order to create an equation analogous to (82) we first notice from equations (75),(86), (89), and (93) that,

$$\begin{aligned} A^T\tilde{Q}_s = A^T Q_m U_s^{(\mathrm{hm})} &= P_m B_m^T U_s^{(\mathrm{hm})} + f e_m^T U_s^{(\mathrm{hm})} \\ &= [P_m\ p_{m+1}] B_{m,m+1}^T U_s^{(\mathrm{hm})} = [P_m\ p_{m+1}]\, \hat{V}_s\, \Sigma_s^{(\mathrm{hm})} \\ &= [P_m\ p_{m+1}] \begin{array}{c|c|c} \hat{V}_s^{(\mathrm{hm})} & & \\ 0 & \hat{v} & \hat{R}_{s+1} \end{array} \begin{bmatrix} I_s \\ e_{m+1}^T \hat{V}_s \end{bmatrix} \Sigma_s^{(\mathrm{hm})} \\ &= [P_m\ p_{m+1}] \left[\begin{array}{c|c} \hat{V}_s^{(\mathrm{hm})} & \\ 0 & \hat{v} \end{array}\right] \hat{R}_{s+1} \begin{bmatrix} \Sigma_s^{(\mathrm{hm})} \\ e_{m+1}^T \hat{V}_s \Sigma_s^{(\mathrm{hm})} \end{bmatrix}. \end{aligned}$$

Next from (94) it follows that $(\tilde{P}_s^T A^T \tilde{Q}_s)^T = \tilde{Q}_s^T A \tilde{P}_s = \tilde{B}_s$ and we define

$$[\rho_1 \dots \rho_s] := e_{s+1}^T \hat{R}_{s+1} \begin{bmatrix} \Sigma_s^{(\mathrm{hm})} \\ e_{m+1}^T \hat{V}_s \Sigma_s^{(\mathrm{hm})} \end{bmatrix}, \quad p_{s+1} := P_m \hat{v}(1\!:\!m) + \hat{v}(m+1) p_{m+1}.$$

Returning back to the expression for $A^T\tilde{Q}_s$ we obtain an equation analogous to (82) as

$$\begin{aligned} A^T\tilde{Q}_s &= [P_m\ p_{m+1}] \left[\begin{array}{c|c} \hat{V}_s^{(\mathrm{hm})} & \\ 0 & \hat{v} \end{array}\right] \begin{bmatrix} \tilde{B}_s^T \\ \rho_1 \dots \rho_s \end{bmatrix} \\ &= [P_m \hat{V}_s^{(\mathrm{hm})}\ p_{s+1}]\, \tilde{B}_{s,s+1}^T =: [\tilde{P}_s\ p_{s+1}]\, \tilde{B}_{s,s+1}^T\,. \end{aligned} \tag{95}$$

Finally, in the spirit of Remark 3.2.1, equations (94)-(95) are also called an s-GKL factorization which can then be extended with a starting vector p_{s+1} via Algorithm 6.2 to obtain an m-GKL factorization like (74)-(75), where the B_m in the factorization will have same structure as in (83) and $\tilde{B}_{s+1}$ is now defined in (95). Then the overall process of computing approximate singular triplets can be continued in the same fashion as for thick–restarted with Ritz vectors, which is outlined in Algorithm 6.3.

3.2.3 GKL and thick–restarted GKL algorithms

We now outline algorithms that implement developments from Sections 3.2.1 and 3.2.2. Algorithm 6.2 determines an m-GKL factorization given either a starting unit vector p_1 or the s-GKL factorizations (81)-(82) or (94)-(95). In Algorithm 6.2 we assume that the m-GKL factorization can be computed, i.e., $\alpha_j > 0$ and $\beta_j > 0$; for a brief discussion on this requirement see [5] and references within. Also note that in order to avoid loss of orthogonality, lines 11 and 16 in Algorithm 6.2 implement reorthogonalization. There are several reorthogonalization strategies cited in the literature, e.g., see [30] for a discussion on partial reorthogonalization or [44] where only the columns of P_m or Q_m are reorthogonalized for matrices that are not too ill–conditioned. Given that our overall scheme in this paper maintains a small value m, in line 16 we implement full reorthogonalization only on the "short" vectors (columns of P_m) for matrices that are not too ill–conditioned and switch to reorthogonalization of columns of Q_m and P_m when it is determined the matrix is ill–conditioned; in Section 3.5 this is referred to as one–sided and two–sided reorthogonalization, respectively.

To denote an approximation to the k desired singular triplets $\{\sigma_j, u_j, v_j\}$ of A from the Krylov subspaces (73), we use notation $\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\}$, where $\sigma_j^{(..)}$, $u_j^{(..)}$, and $v_j^{(..)}$ are taken from the described methods in this paper. For example, based on Section 3.2.1 we can use Ritz values and vectors to write

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{(\mathrm{rz})}, Q_m u_j^{(\mathrm{rz})}, P_m v_j^{(\mathrm{rz})}\}\,. \tag{96}$$

In case of the harmonic Ritz approximations and using $\hat{V}_k^{(\mathrm{hm})}$ from (93), we first compute the k approximations to the singular values

$$\hat{s}_j^{(\mathrm{hm})} \;=\; {u_j^{(\mathrm{hm})}}^T B_m\, \hat{v}_j^{(\mathrm{hm})}, \qquad j = 1, \ldots, k\,, \tag{97}$$

Algorithm 6.2 Golub-Kahan-Lanczos (GKL) Process

Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with A or A^T, m : maximum size of output decomposition,
unit vector p_1 **or** s-GKL factorizations (81)-(82) or (94)-(95).

Output: m-GKL factorizations (74)-(75), with B_m as defined in (76), (83), or (95).

$q_{s+1} := Ap_{s+1}$;

if $s > 0$ **then**
    $q_{s+1} := q_{s+1} - \tilde{Q}_s(\tilde{Q}_s^T q_{s+1})$;
$\alpha_{s+1} := \|q_{s+1}\|$; $q_{s+1} := q_{s+1}/\alpha_{s+1}$;

$Q_{s+1} := [\tilde{Q}_s \; q_{s+1}]$; $P_{s+1} := [\tilde{P}_s \; p_{s+1}]$;

for $j = (s+1) : m$ **do**
    $f := A^T q_j - \alpha_j p_j$;

    Reorthogonalization ("short"): $f := f - P_j(P_j^T f)$;

    if $j < m$ **then**
        $\beta_j := \|f\|$; $p_{j+1} := f/\beta_j$;

        $P_{j+1} := [P_j \; p_{j+1}]$;

        $q_{j+1} := Ap_{j+1} - \beta_j q_j$;

        Reorthogonalization ("long"): $q_{j+1} := q_{j+1} - Q_j(Q_j^T q_{j+1})$;

        $\alpha_{j+1} := \|q_{j+1}\|$; $q_{j+1} := q_{j+1}/\alpha_{j+1}$; $Q_{j+1} := [Q_j \; q_{j+1}]$;

and use

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\hat{s}_j^{(\mathrm{hm})}, Q_m u_j^{(\mathrm{hm})}, P_m \hat{v}_j^{(\mathrm{hm})}\}\,. \tag{98}$$

Remark. *It is important to note the difference in how approximations* (96) *and* (98) *are chosen. By analogy with* (96)*,* $\{\sigma_j^{(hm)}, Q_m u_j^{(hm)}, P_m v_j^{(hm)}\}$ *could have been chosen as an approximation in* (98)*. But that could pose a problem since the singular vector approximations* $\{P_m v_j^{(hm)}\}$ *need not be pairwise orthonormal. Therefore, in cases when more than one singular triplet is desired* ($k > 1$)*, for the final exit of our routine we additionally require that the both sets of vectors* $\{u_j^{(..)}\}_{j=1}^k$ *and* $\{v_j^{(..)}\}_{j=1}^k$ *are orthonormal. We also update* $\sigma_j^{(..)}$*, such that* $\sigma_j^{(..)} = u_j^{(..)^T} B_m v_j^{(..)}$*.*

Convergence in both methods is established by using the following residual equation that is derived from the Lanczos factorization (79) on the augmented matrix $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$,

$$\begin{aligned} resAug_j^{(..)} &= \sqrt{\|AP_m v_j^{(..)} - \sigma_j^{(..)} Q_m u_j^{(..)}\|^2 + \|A^T Q_m u_j^{(..)} - \sigma_j^{(..)} P_m v_j^{(..)}\|^2}\,, \\ &= \sqrt{\|B_m v_j^{(..)} - \sigma_j^{(..)} u_j^{(..)}\|^2 + \|B_m^T u_j^{(..)} - \sigma_j^{(..)} v_j^{(..)}\|^2 + (e_m^T u_j^{(..)})^2 \beta_m^2}\,, \end{aligned} \tag{99}$$

where $\beta_m = \|f\|$. Note that (99) can be simplified when using Ritz approximation (96) to $resAug_j^{(\mathrm{rz})} = e_m^T u_j^{(\mathrm{rz})} \beta_m$. Likewise, a residual equation can be computed from the Lanczos factorization (78) on the normal matrix $A^T A$

$$\begin{aligned} resNor_j^{(..)} &= \|A^T A P_m v_j^{(..)} - (\sigma_j^{(..)})^2 v_j^{(..)}\|\,, \\ &= \sqrt{\|B_m^T B_m v_j^{(..)} - (\sigma_j^{(..)})^2 v_j^{(..)}\|^2 + (\alpha_m e_m^T v_j^{(..)})^2 \beta_m^2}\,. \end{aligned}$$

Note that if $B_m v_j^{(..)} = \sigma_j^{(..)} u_j^{(..)}$, then $resNor_j^{(..)} = \sigma_j^{(..)} resAug_j^{(..)}$. Finally, independent of the restarted scheme used convergence of an approximate triplet is tested via (99) and the condition

$$resAug_j^{(..)} \;\leq\; tol \cdot \|A\|\,, \tag{100}$$

where *tol* is a user specified tolerance and $\|A\|$ is approximated by the largest singular value of B_m over all iterations.

Algorithm 6.3 Thick–restarted GKL with Ritz or harmonic Ritz vectors

Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with A or A^T, m : maximum size of GKL factorization,
k : number of desired singular triplets,
p_1 : unit vector,
tol : tolerance for accepting computed approximate singular triple, cf. (100).

Output: k approximate singular triplets of A $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$.
Compute m-GKL factorization with Algorithm 6.2;

Compute the s-PSVD of B_m (80) with $k \leq s < m$;

Check convergence of k desired triplets (100) with (96);

if thick–restarted with Ritz **then**
    Compute s-GKL factorization (81)-(82);

    Goto 3;

else
    Compute the s-PSVD of $B_{m,m+1}$ (86) ;

    Check convergence (100) with (98);

    Compute s-GKL factorization (94)-(95);

    Goto 3;

3.3 Refined and Iterative Refined Ritz vectors

In 1997, Jia proposed to use *refined Ritz* vectors in place of Ritz vectors as eigenvector approximations of a matrix M [19]. More specifically, for a given approximate eigenvalue μ_j of M, Jia's method looks to minimize $\|Mz_j - \mu_j z_j\|$ for a unit vector

z_j from a given subspace $\mathcal{W}$, i.e.,

$$\min_{z_j \in \mathcal{W},\, \|z_j\|=1} \|Mz_j - \mu_j z_j\|. \tag{101}$$

It was shown that on the subspace $\mathcal{W}$, refined Ritz vectors z_j provided better eigenvector approximations than the Ritz vectors. Moreover, an approximate eigenpair using the refined Ritz vector produced a "smaller" residual norm than an eigenpair approximation with the Ritz pair. Since then, the notion of "refined vectors" has produced a significant amount of research in many directions, see e.g., [2, 16, 17, 20, 21, 22, 23, 24, 25, 26, 27, 29, 37] and the references therein.

More recently, in [2] we introduced the idea of *iterative refined Ritz* values/vectors for the symmetric eigenvalue problem, where the approximate eigenvalue in the refined scheme is replaced with the latest computed refined Ritz value until convergence; for complete details on this development and related theoretical properties see [2]. An important and subtle result regarding iterative refined or refined Ritz vectors is that they are *not* scalar multiples of the same residual vector and as such do not fit naturally into the thick–restarted scheme developed by Wu and Simon [52]. Through numerical examples in [2] we also demonstrated that when memory was limited and only iterative refined Ritz vectors were used to restart the method there was potential for either slow or no convergence. This behavior can, in part, be explained by the refined process (minimization of the norm (101) on a small subspace) possibly favoring the next closest eigenvalue, see [2, Ex. 5.2]. As a way to overcome these challenges, a hybrid method was developed that uses thick–restarted with Ritz vectors and under certain criteria it restarts with a linear combination of iterative refined Ritz vectors. We note that if the iterative refined vectors were replaced with refined vectors in this hybrid scheme, the overall method did not perform as well, see [2, Ex. 5.3 and 6.2].

In this paper, we extend the idea of iterative refined values/vectors to the GKL process and develop a similar hybrid scheme for computing singular triplets. Con-

sidering the relationships of the Lanczos factorizations (78) and (79) and symmetric matrices A^TA and $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$, respectively, the properties in [2] are mostly carried over though some nontrivial adaptations were necessary (e.g., see Section 3.3.2). It is worth noting that refined schemes for computing singular triplets using the Lanczos bidiagonal method with matrix C have been considered in [24, 25]. More specifically, the refined Ritz scheme in [24] uses the lower bidiagonal Lanczos process [40] while the scheme in [25] utilizes the GKL process and computes refined harmonic Ritz values/vectors using the augmented system (79). Both schemes [24, 25] implemented restarting by utilizing the refined process to gain "shifts" that are then used in an implicitly restarted GKL algorithm. Other implicitly restarted GKL methods worth mentioning include [29] where the authors utilized the lower bidiagonal Lanczos process on the related system AA^T while using Ritz or harmonic Ritz values as "shifts", and the method in [6] that used Leja points as "shifts" from the normal equations (78). In contrast to these methods, the primary focus of our method in this paper is not on computing "shifts" but rather on a *hybrid restarting scheme* that restarts the GKL process either through thick–restarting with Ritz/harmonic Ritz or explicitly restarting with linear combination of iterative refined Ritz vectors.

3.3.1 Refined and Iterative Refined on normal system

Our development of the iterative refined Ritz values/vectors naturally starts with the normal system (78). To that end, let $M = A^TA$ and $\mathcal{W} = \mathbb{K}_m(A^TA, p_1)$ in equation (101) and define

$$T_{m+1,m} := \begin{bmatrix} B_m^T B_m \\ \alpha_m \beta_m e_m^T \end{bmatrix} \in \mathbb{R}^{(m+1)\times m}. \tag{102}$$

For each approximate eigenvalue μ_j of A^TA compute the smallest singular value $\sigma_j^{(\text{rf-n})}$ and associated unit singular vectors of $(T_{m+1,m} - \mu_j I_{m+1,m})$, i.e.,

$$(T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{(\text{rf-n})} = \sigma_{\downarrow j}^{(\text{rf-n})} w_j, \tag{103}$$

$$(T_{m+1,m} - \mu_j I_{m+1,m})^T w_j = \sigma_{\downarrow j}^{(\text{rf-n})} v_j^{(\text{rf-n})}, \tag{104}$$

where $v_j^{(\text{rf-n})} \in \mathbb{R}^m$ and $w_j \in \mathbb{R}^{m+1}$. Then from (74),(75), and (78) it follows that

$$\min_{\substack{z_j \in \mathbb{K}_m(A^TA, p_1) \\ \|z_j\|=1}} \|A^TAz_j - \mu_j z_j\| = \|(T_{m+1,m} - \mu_j I_{m+1,m}) v_j^{(\text{rf-n})}\| = \sigma_{\downarrow j}^{(\text{rf-n})} \tag{105}$$

and the *refined Ritz vector* z_j for μ_j is defined as $z_j := P_m v_j^{(\text{rf-n})}$. The approximate eigenvalue of A^TA associated with the refined Ritz vector z_j is selected as the Rayleigh quotient

$$\sigma_j^{(\text{rf-n})^2} = z_j^T A^T A z_j = v_j^{(\text{rf-n})^T} B_m^T B_m v_j^{(\text{rf-n})} = \|B_m v_j^{(\text{rf-n})}\|^2, \tag{106}$$

and the approximate *refined singular triplet on the normal system* for A is given by

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} = \{\sigma_j^{(\text{rf-n})}, Q_m u_j^{(\text{rf-n})}, P_m v_j^{(\text{rf-n})}\}, \tag{107}$$

where $u_j^{(\text{rf-n})} = B_m v_j^{(\text{rf-n})} / \sigma_j^{(\text{rf-n})}$.

The initial approximate eigenvalue μ_j in equations (103)-(105) can be taken as the Ritz value $\sigma_j^{(\text{rz})^2}$ (80). Alternately, μ_j can be chosen as harmonic Ritz value $\sigma_j^{(\text{hm})^2}$ (86) in which case the output from the refined process is referred to as *refined harmonic Ritz*. For the purpose of streamlining exposition, in this section we initially set approximate eigenvalues $\mu_j = \sigma_j^{(\text{rz})^2}$ and refer the reader to [21, 23] for theoretical considerations on refined harmonic Ritz. The iterative refined Ritz process iteratively refines the approximation, by taking the output approximation, $\sigma_j^{(\text{rf-n})}$ (106), setting $\mu_j = \sigma_j^{(\text{rf-n})^2}$, and re–computing refined vectors $v_j^{(\text{rf-n})}$, via (103)-(104) until convergence. This process produces a nonnegative, decreasing and hence convergent sequence $\sigma_{\downarrow j}^{(..)(i)}$, see [2, Thm. 5.1]; Algorithm 6.4 outlines this process.

There are several options for the convergence check (steps 8 and 16) in Algorithm 6.4, e.g., $|\sigma_j^{(..)(i)} - \sigma_j^{(..)(i-1)}|/|\sigma_j^{(..)(i)}| < eps$, where eps is machine epsilon; the

Algorithm 6.4 Iterative Refined

Input: $T_{m+1,m} \in \mathbb{R}^{(m+1)\times m}$ (102) or $T_{2m+1,2m} \in \mathbb{R}^{(2m+1)\times 2m}$ (122) and $\{\mu_j\}_{j=1}^{k}$.

Output: $\{\sigma_j^{(\text{it-n})}, u_j^{(\text{it-n})}, v_j^{(\text{it-n})}\}_{j=1}^{k}$ and $\hat{\sigma}_j^{(\text{it-n})}$ or $\{\sigma_j^{(\text{it-a})}, u_j^{(\text{it-a})}, v_j^{(\text{it-a})}\}_{j=1}^{k}$ and $\hat{\sigma}_j^{(\text{it-a})}$.

for $j = 1, 2, \ldots, k$ **do**
  for $i = 1, 2, \ldots, maxitref$ **do**
    if normal system **then**
      Compute $v_j^{(\text{rf-n})^{(i)}}, w_j^{(i)}$, and $\sigma_j^{(\text{rf-n})^{(i)}}$ (103) and (104);

      $\sigma_j^{(\text{rf-n})^{(i)}} := \|B_m v_j^{(\text{rf-n})^{(i)}}\|$ (106);
      if converge **then**
        $\sigma_j^{(\text{it-n})} := \sigma_j^{(\text{rf-n})^{(i)}}, v_j^{(\text{it-n})} := v_j^{(\text{rf-n})^{(i)}}, u_j^{(\text{it-n})} := B_m v_j^{(\text{it-n})}/\sigma_j^{(\text{it-n})}$, $\hat{\sigma}_j^{(\text{it-n})} := \sigma_j^{(\text{rf-n})^{(i)}}$;

        Break;

      $\mu_j := (\sigma_j^{(\text{rf-n})^{(i)}})^2$;

    else
      Compute $x_j^{(i)}$, $y_j^{(i)}$, $w_{x_j}^{(i)}$, $w_{y_j}^{(i)}$, $w_{z_j}^{(i)}$, and $\sigma_j^{(\text{rf-a})^{(i)}}$ (123) and (124);
      $\sigma_j^{(\text{rf-a})^{(i)}} := 2x_j^{(i)^T} B_m y_j^{(i)}$ (126);
      if converge and $|\|x_j^{(i)}\| - 1/\sqrt{2}| \leq \sqrt{eps}$ **then**
        $\sigma_j^{(\text{it-a})} := \sigma_j^{(\text{rf-a})^{(i)}}$, $v_j^{(\text{it-a})} := y_j{}^{(i)}/\|y_j{}^{(i)}\|$, $u_j^{(\text{it-a})} := x_j^{(i)}/\|x_j^{(i)}\|$, $\hat{\sigma}_j^{(\text{it-a})} := \sigma_j^{(\text{rf-a})^{(i)}}$;

        Break;

      $\mu_j := \sigma_j^{(\text{rf-a})^{(i)}}$;

additional requirement on $\|x_j^{(i)}\|$ in step 16 is discussed in Section 3.3.2. While using finite arithmetic, stagnation can occur and we propose including an additional check to exit when detected. We identify stagnation as failed convergence. The initial view of Algorithm 6.4 (for loop *maxitref*) may appear to be computationally expensive, however when the matrix B_m is kept very small, the cost is negligible in comparison to the cost of the matrix–vector products when the order of A is very large. We include computational times for numerical examples in Section 3.5. When m is larger or as the overall scheme converges, we found that fewer iterations are needed and the iterative refined vectors did not differ much from the refined vectors. However, it should be noted again that the main focus of this paper is on using a very small subspaces, where differences are readily observed. Therefore, using Algorithm 6.4 with initial approximate eigenvalues $\mu_j = {\sigma_j^{(\text{rz})}}^2$, we obtain the approximate *iterative refined Ritz singular triplet on the normal system* for A as

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{(\text{it-n})}, Q_m u_j^{(\text{it-n})}, P_m v_j^{(\text{it-n})}\}. \tag{108}$$

Using the m-GKL factorization and the refined Ritz singular approximation (107), together with equations (103)-(104), give us

$$AP_m v_j^{(\text{rf-n})} \;=\; Q_m B_m v_j^{(\text{rf-n})} \;=\; \sigma_j^{(\text{rf-n})} Q_m u_j^{(\text{rf-n})}, \tag{109}$$

$$\begin{aligned} A^T Q_m u_j^{(\text{rf-n})} &= P_m B_m^T u_j^{(\text{rf-n})} + f e_m^T u_j^{(\text{rf-n})}, \\ &= \sigma_j^{(\text{rf-n})} P_m v_j^{(\text{rf-n})} + \sigma_{\downarrow j}^{(\text{rf-n})} / \sigma_j^{(\text{rf-n})} \big[P_m \; p_{m+1}\big] r_j, \end{aligned} \tag{110}$$

where $r_j = w_j - ([v_j^{(\text{rf-n})}; 0]^T w_j)[v_j^{(\text{rf-n})}; 0]$. Multiplying (109) by A^T on the left yields the following relation

$$A^T A P_m v_j^{(\text{rf-n})} \;=\; {\sigma_j^{(\text{rf-n})}}^2 P_m v_j^{(\text{rf-n})} + \sigma_{\downarrow j}^{(\text{rf-n})} \big[P_m \; p_{m+1}\big] r_j. \tag{111}$$

If Algorithm 6.4 is used to compute the iterative refined Ritz value and vectors we have the output satisfying,

$$(T_{m+1,m} - {\sigma_j^{(\text{it-n})}}^2 I_{m+1,m}) v_j^{(\text{it-n})} = \hat{\sigma}_{\downarrow j}^{(\text{it-n})} \hat{w}_j, \tag{112}$$

$$(T_{m+1,m} - {\sigma_j^{(\text{it-n})}}^2 I_{m+1,m})^T \hat{w}_j = \hat{\sigma}_{\downarrow j}^{(\text{it-n})} v_j^{(\text{it-n})}, \tag{113}$$

and since ${\sigma_j^{(\text{it-n})}}^2 = {v_j^{(\text{it-n})}}^T B_m^T B_m v_j^{(\text{it-n})}$ we have from (112) $[v_j^{(\text{it-n})}; 0]^T \hat{w}_j = 0$. Analogous to equations (109)-(110) with iterative refined Ritz singular approximation (108) we have,

$$AP_m v_j^{(\text{it-n})} = Q_m B_m v_j^{(\text{it-n})} = \sigma_j^{(\text{it-n})} Q_m u_j^{(\text{it-n})} \tag{114}$$

$$\begin{aligned} A^T Q_m u_j^{(\text{it-n})} &= P_m B_m^T u_j^{(\text{it-n})} + f e_m^T u_j^{(\text{it-n})} \\ &= \sigma_j^{(\text{it-n})} P_m v_j^{(\text{it-n})} + \hat{\sigma}_{\downarrow j}^{(\text{it-n})} / \sigma_j^{(\text{it-n})} \big[P_m \; p_{m+1}\big] \hat{w}_j \end{aligned} \tag{115}$$

and after multipling (114) by A^T

$$A^T A P_m v_j^{(\text{it-n})} = {\sigma_j^{(\text{it-n})}}^2 P_m v_j^{(\text{it-n})} + \hat{\sigma}_{\downarrow j}^{(\text{it-n})} \big[P_m \; p_{m+1}\big] \hat{w}_j. \tag{116}$$

Applying [2, Eqns. (5.5) and (5.12)] to Lanczos relationships (111) and (116) shows that

$$\hat{\sigma}_{\downarrow j}^{(\text{it-n})} = resNor_j^{(\text{it-n})} \leq resNor_j^{(\text{rf-n})} \leq resNor_j^{(\text{rz})}. \tag{117}$$

Equation (117) shows that the iterative refined Ritz with respect to the normal residual on the same Krylov subspace, $\mathbb{K}_m(A^T A, p_1)$ are better approximations, however an effective restart process that "improves" the next generated Krylov subspace is still needed. Equations (109)-(111) and (114)-(116) show that the refined Ritz and iterative refined Ritz vectors respectively, are not all multiples of the same residual vector, see [2, Thm. 4.3] in context of Lanczos factorization and the symmetric eigenvalue problem. Therefore the thick–restarted scheme presented in Section 3.2 is not

available. However, one can still explicitly restart the GKL algorithm with a linear combination. We first utilize that the approximations are taken from basis vectors and perform a single iteration of the GKL algorithm that avoids a matrix-vector product with A and A^T as follows.

$$\begin{array}{l} \text{1. Given } \bar{v} = \sum_{j=1}^{k} c_j v_j^{(..)} \text{ set } \beta_0 = \|\bar{v}\| \text{ and } \bar{v} = \bar{v}/\beta_0 \\ \text{2. Let } \bar{u} = B_m \bar{v} \text{ set } \alpha_1 = \|\bar{u}\| \text{ and } \bar{u} = \bar{u}/\alpha_1 \\ \text{3. Set } f = P_m(B_m^T \bar{u} - \alpha_1 \bar{v}) + f e_m^T \bar{u} \text{ and } \beta_1 = \|f\| \\ \text{4. Set } p_1 = P_m \bar{v},\ p_2 = f/\beta_1,\ q_1 = Q_m \bar{u} \end{array} \tag{118}$$

The steps in (118) yield the following 1-GKL factorization

$$Ap_1 = q_1 \alpha_1 \,, \tag{119}$$

$$A^T q_1 = \begin{bmatrix} p_1 \,, \, p_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix}, \tag{120}$$

where Algorithm 6.2 can be restarted with p_2. It is worth noting for $k = 1$ and $\bar{v} = v_1^{(\text{rf-n})}$ or $\bar{v} = v_1^{(\text{it-n})}$, equations (119)-(120) are the same as equations (109)-(110) or (114)-(115), respectively. For $k > 1$ the coefficients c_j in (118) can be chosen several ways and greatly impact convergence. For example for eigenvalue problems, Saad [41] suggests using residual norms, which was also used for the refined Ritz algorithm [19, Alg. 1]. In [2] an alternate approach for iterative refined vectors modeled after Morgan [34] was used to eliminate part of the residual vector as the next Krylov subspace is built. Morgan [34] showed that for Ritz vectors and carefully chosen constants c_j that parts of the residual vector is eliminated when multiplied by A in the next iteration to build out the Krylov subspace, which resulted in the same final subspace as when implementing Sorensen's implicitly restarted method [45]. Unfortunately, this equivalence is not present here, though not all is lost. It turns out that we can still eliminate part of the residual. This requires solving a small $(k-1) \times k$ homogeneous system of equations (121) for coefficients c_j, we refer the reader to [2, Sec. 6] for details.

$$\beta_m \begin{bmatrix} e_m^T v_1^{(\text{it-n})} & \cdots & e_m^T v_k^{(\text{it-n})} \\ \sigma_1^{(\text{it-n})^{2(i-2)}} e_m^T B_m^T B_m v_1^{(\text{it-n})} & \cdots & \sigma_k^{(\text{it-n})^{2(i-2)}} e_m^T B_m^T B_m v_k^{(\text{it-n})} \end{bmatrix} \quad i > 1. \tag{121}$$

3.3.2 Refined and Iterative Refined on augmented system

We now turn our attention to developing notions of refined and iterative refined Ritz values/vectors on the augmented system. We start by letting $M = C$ and $\mathcal{W} = \mathbb{K}_{2m}(C, [0; p_1])$ in equation (101) and define

$$T_{2m+1,2m} := \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \\ \beta_m e_m^T & 0 \end{bmatrix} \in \mathbb{R}^{(2m+1)\times 2m}. \tag{122}$$

For each initial eigenvalue approximation μ_j of C compute the smallest singular value $\sigma_j^{(\text{rf-a})}$ and associated unit singular vectors of $(T_{2m+1,2m} - \mu_j I_{2m+1,2m})$, i.e.,

$$(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix} = \sigma_j^{(\text{rf-a})} \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix}, \tag{123}$$

$$(T_{2m+1,2m} - \mu_j I_{2m+1,2m})^T \begin{bmatrix} w_{x_j} \\ w_{y_j} \\ w_{z_j} \end{bmatrix} = \sigma_j^{(\text{rf-a})} \begin{bmatrix} x_j \\ y_j \end{bmatrix}, \tag{124}$$

where $x_j, y_j, w_{x_j}, w_{y_j} \in \mathbb{R}^m$ and w_{z_j} is a scalar. Then it follows that

$$\min_{\substack{z_j \in \mathbb{K}_{2m}(C,[0;p_1]) \\ \|z_j\|=1}} \|Cz_j - \mu_j z_j\| = \|(T_{2m+1,2m} - \mu_j I_{2m+1,2m}) \begin{bmatrix} x_j \\ y_j \end{bmatrix}\| = \sigma_j^{(\text{rf-a})} \tag{125}$$

and the *refined Ritz vector* z_j for μ_j is defined as $z_j := [\, Q_m x_j \,;\, P_m y_j]$. Analogous to the case of the normal system, the approximate eigenvalue of C associated with refined Ritz vector z_j is selected as the Rayleigh quotient

$$\sigma_j^{(\text{rf-a})} = z_j^T C z_j = \begin{bmatrix} x_j \\ y_j \end{bmatrix}^T \begin{bmatrix} 0 & B_m \\ B_m^T & 0 \end{bmatrix} \begin{bmatrix} x_j \\ y_j \end{bmatrix} = 2 x_j^T B_m y_j \,, \tag{126}$$

and the approximate *refined singular triplet on the augmented system* for A is given by

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} = \{\sigma_j^{(\text{rf-a})}, Q_m u_j^{(\text{rf-a})}, P_m v_j^{(\text{rf-a})}\}\,, \tag{127}$$

where $u_j^{(\text{rf-a})} = x_j/\|x_j\|$ and $v_j^{(\text{rf-a})} = y_j/\|y_j\|$. Similar to (111) for the normal system, but this time applied to the Lanczos factorization (79) for the augmented matrix C

we have the following relationship,

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m x_j \\ P_m y_j \end{bmatrix} = \sigma_j^{(\text{rf-a})} \begin{bmatrix} Q_m x_j \\ P_m y_j \end{bmatrix} + \sigma_{\downarrow j}^{(\text{rf-a})} \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} r_{xj} \\ r_{y_j} \\ r_{zj} \end{bmatrix}, \tag{128}$$

where $r_{zj} = w_{z_j}$ is a scalar, $r_{y_j} = w_{y_j} - [x_j\,;\, y_j]^T[w_{xj}\,;\, w_{y_j}]y_j \in \mathbb{R}^m$, and $r_{xj} = w_{xj} - [x_j\,;\, y_j]^T[w_{xj}\,;\, w_{y_j}]x_j \in \mathbb{R}^m$. Given the relationship between the eigenvalues of C and the singular values of A, we can start Algorithm 6.4 with the initial approximation μ_j in equations (123)-(125) as the Ritz value $\sigma_j^{(\text{rz})}$. This now gives us an approximate *iterative refined Ritz singular triplet on the augmented system* for A as

$$\{\sigma_j^{(..)}, Q_m u_j^{(..)}, P_m v_j^{(..)}\} \;=\; \{\sigma_j^{(\text{it-a})}, Q_m u_j^{(\text{it-a})}, P_m v_j^{(\text{it-a})}\}. \tag{129}$$

For convenience, consider the unscaled output vectors of $u_j^{(\text{it-a})}$ and $v_j^{(\text{it-a})}$ from Algorithm 6.4 as the last iteration vectors $\hat{x}_j := x_j^{(i)}$ and $\hat{y}_j := y_j^{(i)}$, respectively. Therefore, analogous to (112)-(113) and (116) we have the output from Algorithm 6.4 that satisfies

$$(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m}) \begin{bmatrix} \hat{x}_j \\ \hat{y}_j \end{bmatrix} \;=\; \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix} \tag{130}$$

$$(T_{2m+1,2m} - \sigma_j^{(\text{it-a})} I_{2m+1,2m})^T \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix} \;=\; \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} \hat{x}_j \\ \hat{y}_j \end{bmatrix}, \tag{131}$$

where $[\hat{x}_j\,;\hat{y}_j]^T\ [\hat{w}_{x_j}\,;\hat{w}_{y_j}] = 0,\ \hat{x}_j, \hat{y}_j, \hat{w}_{x_j}, \hat{w}_{y_j} \in \mathbb{R}^m$, and $\hat{w}_{z_j}$ is a scalar and when applied to the Lanczos factorization (79) gives us the following

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} Q_m \hat{x}_j \\ P_m \hat{y}_j \end{bmatrix} = \sigma_j^{(\text{it-a})} \begin{bmatrix} Q_m \hat{x}_j \\ P_m \hat{y}_j \end{bmatrix} + \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \begin{bmatrix} Q_m & 0 & 0 \\ 0 & P_m & p_{m+1} \end{bmatrix} \begin{bmatrix} \hat{w}_{x_j} \\ \hat{w}_{y_j} \\ \hat{w}_{z_j} \end{bmatrix}. \tag{132}$$

Similar to (117), the relationships (128) and (132) together with [2, Eqns. (5.5) and (5.12)] applied to symmetric matrix C imply that

$$\hat{\sigma}_{\downarrow j}^{(\text{it-a})} = resAug_j^{(\text{it-a})} \leq resAug_j^{(\text{rf-a})} \leq resAug_j^{(\text{rz})}. \tag{133}$$

Equation (133) shows that the iterative refined Ritz with respect to the augmented residual on the same Krylov subspace $\mathbb{K}_{2m}(C,[0;p_1])$ are better approximations. But relation (133) is derived with respect to the unscaled vectors $x_j, y_j, \hat{x}_j, \hat{y}_j$. Unlike the singular vectors computed from the eigenvectors of C, the norms $\|x_j\|$, $\|y_j\|$ $\|\hat{x}_j\|$, and $\|\hat{y}_j\|$ are not necessarily equal to the common value $1/\sqrt{2}$, especially during the onset of the overall routine. However, these norms do approach $1/\sqrt{2}$ as approximations improve and we use it as a part of a convergence requirement in Algorithm 6.4. This requirement is reasonable by observing that from the iterative process of Algorithm 6.4 and equations (122), (123), and (126) it follows that

$$x_j^{(i)} = 1/\sigma_j^{(\text{rf-a})^{(i-1)}} \left(B_m y_j^{(i)} - \sigma_{\downarrow j}^{(\text{rf-a})^{(i)}} w_{x_j}^{(i)} \right). \tag{134}$$

When the iterative refine process converges and $\hat{x}_j := x_j^{(i)}$, then we have $\sigma_j^{(\text{rf-a})^{(i-1)}} = \sigma_j^{(\text{rf-a})^{(i)}} = \sigma_j^{(\text{it-a})} = 2\hat{x}_j^T B_m \hat{y}_j$ and

$$\hat{x}_j = 1/\sigma_j^{(\text{it-a})} \left(B_m \hat{y}_j - \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \hat{w}_{x_j} \right), \tag{135}$$

$$\|\hat{x}_j\|^2 = 1/2 - \hat{\sigma}_{\downarrow j}^{(\text{it-a})} / \sigma_j^{(\text{it-a})} \hat{x}_j^T \hat{w}_{x_j}. \tag{136}$$

If $\hat{\sigma}_{\downarrow j}^{(\text{it-a})} = 0$, then we have the desired property and convergence (see (133)). If $\hat{\sigma}_{\downarrow j}^{(\text{it-a})} \neq 0$, then from (123) and (126) we have the relationship $\hat{x}_j^T \hat{w}_{x_j} = -\hat{y}_j^T \hat{w}_{y_j}$. After multiplying (130) by $[\hat{w}_{x_j}\,;\,0\,;\,0]^T$ and using $B_m \hat{w}_{x_j} - \sigma_j^{(\text{it-a})} \hat{w}_{y_j} = \hat{\sigma}_{\downarrow j}^{(\text{it-a})} \hat{y}_j$ from (131), we obtain

$$\begin{aligned} |\|\hat{x}_j\|^2 - 1/2| &= (\hat{\sigma}_{\downarrow j}^{(\text{it-a})} / \sigma_j^{(\text{it-a})})^2 |\|\hat{w}_{x_j}\|^2 - \|\hat{y}_j\|^2|/2, \\ &\leq (\hat{\sigma}_{\downarrow j}^{(\text{it-a})} / \sigma_j^{(\text{it-a})})^2, \end{aligned} \tag{137}$$

where the inequality is established using the triangle inequality and the fact that $\|\hat{w}_{x_j}\| < 1$ and $\|\hat{y}_j\| < 1$. Through numerical examples, we have found that including $|\|x_j^{(i)}\| - 1/\sqrt{2}| \leq \sqrt{eps}$ with the convergence test in Step 16 in Algorithm 6.4 resulted in a better performance in our hybrid algorithm for the augmented system.

Remark. *We make the following observation from an asymptotic point of view of the iterative refined Ritz values/vectors on the augmented system. As the overall routine converges, it is expected for* $\hat{\sigma}_{\downarrow j}^{(it\text{-}a)}$ *in* (132) *to approach* 0. *As* $\hat{\sigma}_{\downarrow j}^{(it\text{-}a)} \to 0$, *from* (135)-(137) *we have that* $\|\hat{x}_j\| \approx \|\hat{y}_j\| \approx 1/\sqrt{2}$, $u_j^{(it\text{-}a)} \approx 1/\sigma_j^{(it\text{-}a)} B_m v_j^{(it\text{-}a)}$, *and* $\sigma_j^{(it\text{-}a)} \approx \|B_m v_j^{(it\text{-}a)}\|$. *Moreover, we start to see the residual relation* (133) *holding on the normalized vectors and the alignment with the iterative refined Ritz values/vectors on the normal system. Therefore, we use formulas* (118) *with* $v_j^{(..)} := v_j^{(it\text{-}a)}$ *to obtain the* 1*-GKL factorization* (119)-(120) *where Algorithm 6.2 can be restarted with* p_2. *Likewise, when* $k > 1$, *we can replace* $v_j^{(..)} := v_j^{(it\text{-}a)}$ *and* $\sigma_j^{(..)} := \sigma_j^{(it\text{-}a)}$ *and solve the homogeneous system* (121) *to restart with a linear combination of vectors. Although an alignment is eventually expected, there are convergence differences, see the numerical examples in Section 3.5.*

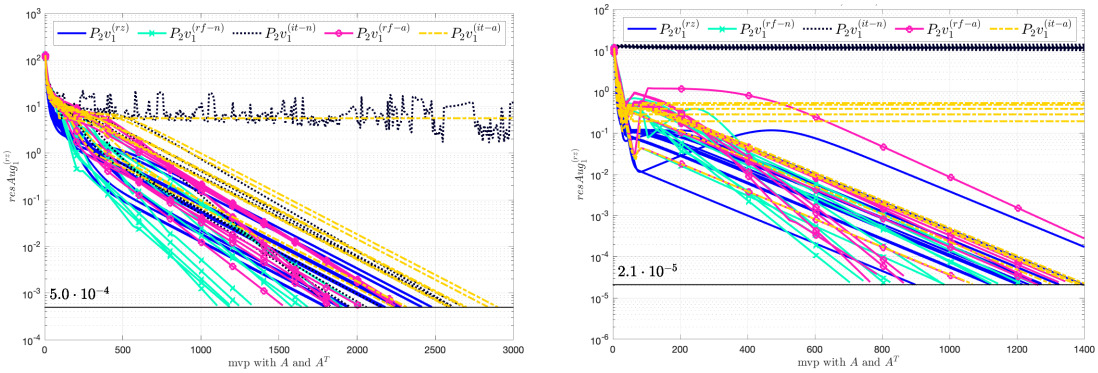
The goal of the paper is the development of a restarted GKL scheme with a focus on keeping the value m relatively small in relationship with k, e.g. $m = k + 1$. Even though the refined and iterative refined values/vectors yield a "smaller" residual norm on the same Krylov subspace than Ritz values/vectors, restarting with these "better" vectors in presence of small m value may not always yield a "better" Krylov subspace on the next iteration. Putting aside the efficiency of thick–restarted when $k > 1$ and focusing on restarting with a single vector, we see that the minimization process may not always coincide with the best restart vector. Even when refined vectors do yield better results, the results may not be profoundly significant. The following example illustrates this point.

Example 3.3.1 For this example, we look at two matrices, $A = \text{diag}(1:500)$ a 500×500 diagonal matrix and A being the 262111×262111 amazon0302 matrix from the SuiteSparse Matrix Collection [10]. We let $k = 1$ and $m = 2$ and search for the largest singular triplet with tolerance 10^{-6} while using (100) as a stopping criteria.

We started Algorithm 6.2 with a random vector p_1 and then on the next restart of Algorithm 6.2 we computed p_1 to be Ritz vector $P_m v_1^{(\text{rz})}$, refined Ritz on normal system $P_m v_1^{(\text{rf-n})}$, iterative refined Ritz on normal system $P_m v_1^{(\text{it-n})}$, refined Ritz on augmented system $P_m v_1^{(\text{rf-a})}$, or iterative refined Ritz on augmented system $P_m v_1^{(\text{it-a})}$. All methods for both examples ran 10 times with a different random starting vector p_1 where each method started with the same random vector. Since this example is focused on measuring the overall convergence, i.e., the quality of the Krylov subspace, and for ease of comparison, we only computed the common Ritz norm for each restart method, i.e., $resAug_1^{(\text{rz})}$. The results are presented in Figure 3. The figures display the number of matrix vector products with A and A^T against the residual norm computed with Ritz value $\sigma_1^{(\text{rz})}$ and vector $P_m v_1^{(\text{rz})}$, i.e., $resAug_1^{(\text{rz})}$ for all routines.

We see from the outputs in Figure 3 a wide range of convergence with typically (not always) restarting with refined Ritz vectors performing better than restarting with Ritz vectors. Figure 3 also shows the methods struggling at the beginning, especially with the amazon0302 example (see Figure 3b). This suggest that the methods are having difficulty on a small subspace, capturing the needed components of desired singular vector for restarting. Although not displayed, and as expected, when we increased the value of m the differences between routines becomes smaller with all routines converging, e.g., for the diagonal matrix, when $m = 10$ all routines converged between about 300 and 380 matrix–vector products.

The calculations of iterative refined Ritz vectors are more sensitive to converging to the next closest Ritz value during the iteration process causing stagnation. Figure 3b shows iterative refined Ritz on the normal system stagnating for all restarts. At first glance, this sensitivity appears to be a disadvantage for the iterative refined vectors, but it can in fact be used to more easily signal when the iterative refined vectors should not be used to restart the system. Exactly this reliance on sensitivity



(a) $A = \text{diag}(1:500)$ (b) $A =$ amazon0302 from [10]

Figure 3. Example 3.3.1. Each line represents a start with a random vector and then a restart using the stated vector in the legend. See Example 3.4.1 for same systems with Algorithm 6.5.

is what led us to develop a new hybrid method in this paper. Our hybrid method uses thick–restarting with either Ritz or harmonic Ritz vectors and when certain criteria are met it switches to restarting with iterative refined Ritz vectors on the normal or the augmented system. This combination works very well when m is relatively small – for relevant discussion and numerical results on the same matrices but with our hybrid method Algorithm 6.5, see Example 3.4.1 and Figure 4.

3.4 Hybrid Iterative Refined Algorithms

We now present the main contributions of this paper, namely Algorithms 6.5–6.6. In the previous section, we saw that despite the fact that the iterative refined Ritz values/vectors can provide "better" approximations on a given Krylov subspace, using them to restart the GKL method need not yield better overall convergence and in fact can lead to stagnation (see Example 3.3.1). The stagnation or slow convergence is contributed in part to the use of a small subspace along with the iterative refined process causing the output approximation to be "closer" to the next closest Ritz value/vector. This apparent weakness turns out to be more of an advantage of the iterative refined Ritz values/vectors, since they can be used to more easily identify

when they are not a good choice to restart the GKL process. The implications of this are twofold: there ought to be a process to determine when the iterative refined vectors are good to be used to restart the GKL process and an alternate choice of restart vectors is needed. The thick–restarted methods described in Section 3.2 are a more efficient restarted GKL method with a theoretical connection to implicit restarting, but can not be used directly with (iterative) refined Ritz vectors. Consequently, we advocate to use a hybrid method that utilizes thick–restarted as the main routine and, under certain conditions described below, switches to restarting with iterative refined Ritz vectors.

The parameters to switch between thick–restarting and restarting with iterative refined vectors were chosen based on numerous experiments across a variety of problems. A careful balance is needed, since on the one side the iterative refined Ritz vectors can give a better approximation but with possible stagnation, while on the other side thick–restarted is a more efficient restarting scheme, but with not as good of approximations. Therefore, we first check the angle via the inner product between the desired iterative refined vector and the Ritz vector to determine that the refined process did not cause the vectors to deviate too far from each other. If the angle is acceptable, we use iterative refined Ritz vector(s) to restart. Numerous experiments suggest using

$$\min_{1\leq j\leq k} |{v_j^{(\mathrm{rz})}}^T v_j^{(..)}| \; > \; 0.9\,, \tag{138}$$

where $v_j^{(..)} := v_j^{(\text{it-n})}$ if using the normal system and $v_j^{(..)} := v_j^{(\text{it-a})}$ if using the augmented system. Although we have not encountered the following situation in practice, it is worth noting that it is possible that a Ritz vector may not have any accuracy from the same subspace even though the refined vector is arbitrarily close to the desired eigenvector, see [22, 27]. Since thick–restarted is the main routine with theoretical connection to implicitly restarting techniques and foundations for publicly available

software, it is reasonable to assume that as the sequence of generated Krylov subspaces changes on each new iteration that the Ritz approximations will also change and improve.

Secondly, in order to ensure convergence and avoid missing singular triplets ($k > 1$), we also require the input value μ_j into Algorithm 6.4 to be the best approximation for singular value of A over all computed $\sigma_j^{(\text{rz})}$'s values thus far and to reject using restarting with iterative refined Ritz vectors if the current computed iterative refined values, $\sigma_j^{(\text{it-n})}$ or $\sigma_j^{(\text{it-a})}$, are not "better" than the past iteration's best approximation. For example, during a current iteration (iter) of Algorithm 6.5 we require in step 9 for the call to Algorithm 6.4 that

$$\mu_j \quad = \quad \max_{1 \le \text{i} \le \text{iter}} |\sigma_j^{(\text{rz})^{(i)}}| \quad \text{for} \quad 1 \le j \le k \tag{139}$$

and for step 10

$$|\sigma_j^{(..)^{(\text{iter})}}| \quad \geq \quad \max_{1 \le \text{i} \le \text{iter-1}} |\sigma_j^{(\text{rz})^{(i)}}| \quad \text{for} \quad 1 \le j \le k\,, \tag{140}$$

where $\sigma_j^{(..)} := \sigma_j^{(\text{it-n})}$ for normal system and $\sigma_j^{(..)} := \sigma_j^{(\text{it-a})}$ for augmented system. When $k = 1$ we found that using (139) was a needed requirement for the best results, but encountered poor convergence results when enforcing (140) with $m = 2$. Similar criteria is used for searching for the smallest singular triplets. Additionally, due to a negligible computational cost, various convergence checks are performed at different stages of Algorithm 6.5, e.g., see steps 4, 7, 11, and 17 – this allows for Algorithm 6.5 to exit at the right time and to avoid performing unnecessary expensive computations.

We note to the reader that Algorithm 6.5 is a simplification of the actual computations performed. For instance, in the thick–restarted steps 19 and 21 in Algorithm 6.5 we compute s-GKL factorization where $s \geq k$ to restart Algorithm 6.3. The technique of including additional vectors ($> k$) is a very common strategy to accelerate the convergence in restarted methods. Similarly a gap strategy can also

be used to accelerate the convergence by avoiding shifting too close to the desired spectrum. For example, in the implicitly shifted Lanczos bidiagonalization schemes, a relative gap strategy can be used to enhance convergence, see [7, 24, 25, 30] for details. Considering the connection between implicitly shifting with (harmonic) Ritz and thick–restarting, a simple gap strategy can also be used when deciding on adding additional vectors. We implemented the following straightforward and effective strategy for choosing $s \geq k$,

$$\begin{array}{l} s = k + nc; \\ \text{if } \; |\sigma_{s+1} - \sigma_s| < |\sigma_s - \sigma_{s-1}|, \; s = s + 1; \; \text{end} \\ s = \max(\text{floor}((m + nc)/2), s); \\ \text{if } s >= m, \; s = m - 1; \; \text{end} \end{array} \tag{141}$$

where n_c is the number of converged singular triplets, see [51] for details and comparison of techniques. The strategy in (141) works well in this context, particularly when difference between k and m is kept relatively small. When restarting with iterative refined Ritz vectors, relations (141) were too aggressive and rarely satisfied the requirements (138) and (140) for all $s > k$ and therefore we always use k iterative refined Ritz vectors for restarting. However, using k iterative refined Ritz vectors to restart can cause an unfortunate increase in the residual norms measured by Ritz values/vectors, particularly when $k > 1$. This can be seen in part as negating the idea of the gap strategy mentioned above. Consequently, we do not restart consecutively with iterative refined Ritz vectors if the last restart with iterative refined Ritz vectors caused the norm of Ritz vectors/values to increase from the previous iteration.

Example 3.4.1 We now revisit Example 3.3.1 and still consider the same matrices $A = \text{diag}(1:500)$ and $A =$ amazon0302. We continue to use the same values as specified in Example 3.3.1, namely $k = 1$, $m = 2$, and tolerance 10^{-6}. But now we use Algorithm 6.5 on two hybrid methods, restarting with $P_m v_1^{(\text{rz})}$ and $P_m v_1^{(\text{it-n})}$ (iterative refined Ritz on normal system) and $P_m v_1^{(\text{rz})}$ and $P_m v_1^{(\text{it-a})}$ (iterative refined Ritz on augmented system). Similar to Example 3.3.1, we ran all methods in both

Algorithm 6.5 Hybrid: Thick—Restarted – Restarted SVDS `(trrsvds)`

Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with A or A^T, m : maximum size of GKL factorization,
k : number of desired singular triplets,
p_1 : unit vector,
tol : tolerance for accepting computed approximate singular triple, cf. (100).

Output: k approximate singular triples $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$ of A.
Compute m-GKL factorization with Algorithm 6.2;

Compute the SVD of B_m (80) and check $1 \le j \le k$ (100) with (96);

if thick-restarted with harmonic Ritz **then**
    Compute the s-PSVD of $B_{m,m+1}$ (86) where $k \le s < m$;

    Check $1 \le j \le k$ (100) with (98);

Compute $\{\sigma_j^{(..)}, u_j^{(..)}, v_j^{(..)}\}_{j=1}^{k}$ by Algorithm 6.4 with μ_j (139) for either the augmented system or the normal system;

if all $\sigma_j^{(..)}$ converged and satisfy (138) and (140) **then**
    Check $1 \le j \le k$ (100) with (108) or (129);
    if $k > 1$ **then**
        Compute c_j from (121);

    Compute 1-GKL factorization (119)-(120);
else
    Check $1 \le j \le k$ (100) with (127) and μ_j (139);

    if thick–restarted with Ritz **then**
        Compute s-GKL factorization (81)-(82) where $k \le s < m$;

    else
        Compute s-GKL factorization (94)-(95) where $k \le s < m$;

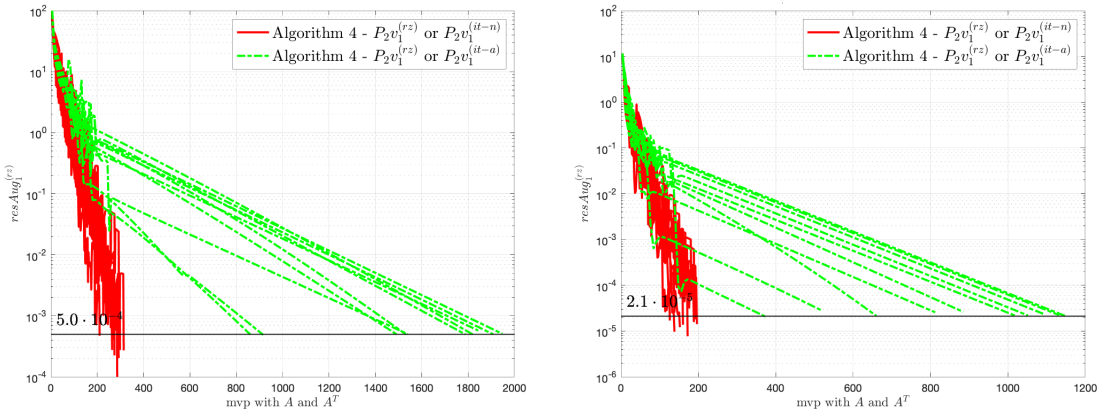
Goto 3;

examples, 10 times with a different random starting vector p_1 where each method started with a same random vector.

In Figure 4 we collect the results, where the graphs display the number of matrix vector products with A and A^T against the residual norm computed with Ritz value $\sigma_1^{(\mathrm{rz})}$ and vector $P_m v_1^{(\mathrm{rz})}$, i.e., $resAug_1^{(\mathrm{rz})}$ for all routines. More specifically, for the diagonal system, Figure 4a shows that our hybrid method with iterative refined Ritz on normal system always converged between 210 and 315 matrix–vector products with respect to $resAug_1^{(\mathrm{rz})}$, compared to Example 3.3.1 where the best result for diagonal matrix is 1100 matrix-vector products. Similarly, for amazon0302 matrix, Figure 4b shows the hybrid method with iterative refined Ritz on normal system always converged between 125 and 205 matrix–vector products with respect to $resAug_1^{(\mathrm{rz})}$ while comparable computation in Example 3.3.1 required about 700 matrix-vector products. This clearly illustrates that Algorithm 6.5 restarting with $P_m v_1^{(\mathrm{rz})}$ and $P_m v_1^{(\text{it-n})}$ performed significantly better than all restarted methods in Example 3.3.1. Furthermore, we emphasize that in comparison to Example 3.3.1 Algorithm 6.5 avoided stagnation which was one of the motivating factors for its development.

Remark. *We note that in the context of Example 3.4.1, if iterative refined Ritz vectors were replaced with refined Ritz vectors in Algorithm 6.5, then we saw almost no performance increases over the results in Example 3.3.1 for restarting with refined Ritz vectors. This is attributed in part to the angle criteria* (138) *for switching being almost always satisfied, a similar observation was made in the context of eigenvalue computations in [2, Examples 5.3 and 6.2].*

We conclude this section with a discussion of our second hybrid scheme, namely Algorithm 6.6, which can be viewed as a simple yet powerful variant of Algorithm 6.5. The main difference between these two hybrid schemes is the way they treat the case when multiple singular triplets are desired, i.e., $k > 1$. More specifically, Algo-



(a) $A = \text{diag}(1:500)$ (b) $A =$ amazon0302 from [10]

Figure 4. Example 3.4.1: Each line represents a start with a random vector and then a restart using the stated vector in the legend. See Example 3.3.1 for same systems using non-hybrid restarting methods.

rithm 6.6 uses a deflation technique that requires the generated basis vectors for the Krylov subspaces to be orthogonal to the converged singular vectors. We do not advocate the inclusion of this technique in Algorithm 6.5 because Algorithm 6.5 is set up as a method that continuously updates the singular vectors regardless of their residual values. This process can be referred to as "soft" locking while the deflation process in Algorithm 6.6 can be referred to as "hard" locking – for more details on "soft" and "hard" locking we direct the reader to [46].

In case when $k > 1$, a typical implementation of a deflation technique searches for k singular triplets and as a singular triplet converges, "locks" the singular vectors, reduces the search space to avoid increasing memory, and orthogonalizes the subsequent generated basis vectors against the converged singular vectors, see [42] for details. Note however that if such an approach was to be implemented in Algorithm 6.5, then it would add "hard" locking to a routine which already effectively utilizes "soft" locking. Therefore, considering the compelling results in Example 3.4.1 for the normal system and a very small basis size, we deviate from a typical imple-

mentation and instead fix $k = 1$ and $m = 2$ and as a singular triplet converges, "lock" the singular vectors and orthogonalize the subsequent generated basis vectors against the converged singular vectors. The "large" memory requirement for such deflation as implemented in Algorithm 6.6, is only five vectors (p_1, p_2, q_1, q_2, f) and the $2(k-1)$ converged singular vectors. Notice that increasing $m > 2$, and the remarks on accelerating the convergence along with formula (141), would revert our implementation back to including thick–restarted and mixing locking techniques.

There are clear advantages to fixing $k = 1$ and $m = 2$, namely reducing the overall computational cost beyond matrix-vector products, e.g., for a majority of problems one does not need to reorthgonalize the basis vectors, though for very ill-conditioned problems a second orthgonalization is recommended. Moreover, the resulting algorithm is a simple yet powerful routine which is easy to understand and implement ($\approx$ 100 lines of MATLAB code).

It should be noted though that if the space spanned by converged vectors is not computed accurately enough, then orthogonalizing against it could slow down the overall convergence. Therefore, a tolerance used to deflate converged vectors should be better than the user input desired tolerance for the first $(k-1)$ singular triplets. Our numerical experiments and previous experiences, led us to set the tolerance for deflation to be the minimum between $\sqrt{eps}$ and $10^{-2} \cdot tol$, where tol is the user input tolerance. When $k > 1$ and convergence is determined for deflation, we restart the method with the residual vector f from the last iteration, orthogonalized against the converged right singular vector(s) and then proceed to orthogonalize all subsequent computed basis vectors against the converged singular vector(s), see step 4 in Algorithm 6.6.

Algorithm 6.6 Hybrid: Restarted Deflation (2×2) SVDS `(rd2svds)`

Input: $A \in \mathbb{R}^{\ell \times n}$ or functions for evaluating matrix-vector products with A or A^T, k : number of desired singular triplets,
p_1 : unit vector,
tol : tolerance for accepting computed approximate singular triple, cf. (100).

Output: k approximate singular triples $\{\sigma_j, u_j, v_j\}_{j=1}^{k}$ of A.

$j := 1$;
Compute 2-GKL factorization with Algorithm 6.2 where for $i = 1, 2, \ldots, (j-1)$ $P_2^T v_i = 0$, $f^T v_i = 0$, and $Q_2^T u_i = 0$;

Compute the largest singular triplet $\{\sigma_1^{(\text{rz})}, u_1^{(\text{rz})}, v_1^{(\text{rz})}\}$ of B_2 (80);

Compute $\{\sigma_1^{(\text{it-n})}, u_1^{(\text{it-n})}, v_1^{(\text{it-n})}\}$ by Algorithm 6.4 with μ_1 (139);

if $j < k$ and (100) is satisfied with $tol = \min\{\sqrt{eps}, 10^{-2} tol\}$ using either (108) or (127) with μ_1 (139); **then**
    $\{\sigma_j, u_j, v_j\} := \{\sigma_1^{(\text{it-n})}, Q_2 u_1^{(\text{it-n})}, P_2 v_1^{(\text{it-n})}\}$ or $\{\sigma_j, u_j, v_j\} := \{\sigma_1^{(\text{rf-a})}, Q_2 u_1^{(\text{rf-a})}, P_2 v_1^{(\text{rf-a})}\}$;
    Compute $f = f - (v_j^T f) v_j$;
    $p_1 := f/\|f\|$, $j := j + 1$;
    Goto 4;
else
    Check (100) with (108) or (127) and μ_1 (139);
if $\sigma_1^{(\text{it-n})}$ converged and satisfies (138) **then**
    Compute 1-GKL factorization (119)-(120);
else
    Compute 1-GKL factorization (81)-(82);

Goto 4;

3.5 Numerical Examples

In this section, we present MATLAB codes `trrsvds`[2] and `rd2svds`[2] which implement Algorithm 6.5 and Algorithm 6.6, respectively. The numerical experiments in this section are specifically created to provide the reader with an insight into the developed methods and illustrate their performance. To that end, we compare our methods to five other routines: two publicly available MATLAB codes `irlba` [5][3] and `svdifp` [33][4], a publicly available MATLAB interfaced code `primme_svds`[53][5], and MATLAB's built–in functions `svds` and `eigs`, where `eigs` is applied to the matrix $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$ and the equivalent eigenvalue problem (79).

Table 4 presents the parameters and default values used in `trrsvds`. Setting parameter *method* = 'thk' results in `trrsvds` implementing only thick–restarting with Ritz or harmonic Ritz vectors, i.e., Algorithm 6.3. In essence this is very similar to `irlba`, though there are a some noted differences. Both methods include additional vectors as way of improving convergence, but while our method uses a dynamic scheme (141) `irlba` has the parameter *adjust* which is by default set at three. Furthermore, the `irlba` also increases the number of restart vectors by the number of converged singular triplets. Because of this rigidity of parameter *adjust* and the fact that in all our examples when searching for the largest singular triplets the Lanczos basis is restricted to be as small as possible, we set *adjust* to zero instead of its default value three.

For the parameter choices for the other codes, we refer the reader to the references, code documentation, and noted websites for full details and descriptions of parameters. Here, we only provide remarks on other routines' parameters as needed for clarification. There are numerous selections and variety of combinations of pa-

[2]Code available at: `http://www.math.uri.edu/~jbaglama`

[3]Code available: `http://www.netlib.org/numeralgo/na26.tgz` or `http://www.math.uri.edu/~jbaglama`

[4]Code available at: `https://github.com/wildstone/SVDIFP`

[5]Code available at: `https://github.com/primme/primme`

Table 4. The user specific parameters for `trrsvds`.

roh	Four letter string ('ritz' or 'harm') specifying the use of either Ritz vectors or harmonic Ritz vectors for thick–restarting. Default value: roh ='harm' if $sigma$ ='SS' and roh ='ritz' if $sigma$ ='LS'.
k	Number of desired singular values. Default value: $k = 1$.
m	Number of Lanczos vectors, i.e., the size of bidiagonal Lanczos matrix B_m. Default value: $m = 2$ if $sigma$ ='LS' and $m = 15$ if $sigma$ ='SS' .
$maxit$	Maximum number of restarts. Default value: $maxit = 2000$.
$maxitref$	Maximum number of iterations used to find iterative refined Ritz singular values, see Algorithm 6.4. Default value: $maxitref = 100$
$method$	Three letter string ('nor', 'aug', or 'thk') to determine which method to use. Default value: $method$ ='nor'.
$reorth$	Three letter string ('one' or 'two') specifying whether to use one-sided ('one') or two sided ('two') full reorthogonalization. Default value: $reorth$ ='one'.
$sigma$	Two letter string ('LS' or ' SS') specifying the location of the desired singular values. 'LS' for the largest singular values and 'SS' for the smallest singular values. Default value: $sigma$ ='LS'
tol	Tolerance used for convergence. Convergence is determined by equation (100). Default value: $tol = \sqrt{eps}$ (roughly 10^{-8}), where eps is machine precision.
p_1	An initial starting vector. If $\ell > n$ and $sigma$ ='SS' then $p_1 \in \mathbb{R}^\ell$ else $p_1 \in \mathbb{R}^n$. Default value: $p_1 = randn(n, 1)$.

rameters for each code. Some choices and combinations may yield faster convergence than others. We cannot provide examples with all possible combinations and only present comparisons to illustrate that the developed methods are competitive. For comparison codes we used either the default values for the parameters or parameter choices that represent the fairest comparison with respect to our proposed methods. For all codes, we set the following common parameters: number of desired singular triplets k, a common random starting vector p_1, tolerance tol, and Lanczos basis maximum size m. Instead of a starting vector, routines `svdifp` and `primme_svds` use an input matrix whose ith column is the ith initial approximate right singular vector. Therefore, for those routines we set the first column to be the common starting vector p_1.

In regards to the other parameters, we set the tolerance for `svdifp` to be $tol \cdot \|A\|_2$. This parameter choice provided the same order of magnitude of the residuals computed by `svdifp` as well as the other routines, see Table 6 and captions in Figures 7-17. With respect to a common basis size similar to m in `trrsvds`, we identify the parameter in the other methods that represent the "storage" or basis size. Depending on a routine and a coding style, this parameter may be restricted (e.g, `eigs(C)` and `svds` require $m \geq k + 2$) or additional storage may be included for calculations. We assume that for all methods the parameter that represents "storage" is comparable to the basis size m in `ttrsvds` and is therefore represented by m in Figures 7-17. However, given the complexities and propriety of some of the codes this may not always be the case.

Remark. *We now briefly discuss the required storage of our hybrid method* `rd2svds` *and how it compares to the basis parameter* m *in* `ttrsvds` *and how it is represented in the figures. Recall that for any* $k \leq m$, `ttrsvds` *with basis size* m *requires storage of* $2m + 1$ *vectors. On the other hand,* `rd2svds` *always works with the fixed basis size*

of two and thus it only needs to store $5+2(k-1)$ *vectors, namely,* p_1, p_2, q_1, q_2, f *and* $2(k-1)$ *converged singular vectors. Therefore, when computing* k *singular triplets* `rd2svds` *is only reported for size* $m = k+1$ *in figures, since the number of stored vectors by* `trrsvds` *and* `r2svds` *is the same.*

The program `svdifp` allows application of a preconditioner and can perform better when one is applied, see [33] for details. However, the use of a preconditioner significantly increases the overall storage requirements, counter to this paper's primary goal of using as little storage as possible. Consequently, we did not apply a preconditioner. It should be noted, as stated by the authors of [33], "that `svdifp` without preconditioning is simply the restarted Lanczos method with the LOBPCG-type subspace enhancement."

The MATLAB interfaced code `primme_svds` is part of a massive high performance C99 library PRIMME for computing eigenpairs and singular triplets. There are numerous parameter settings, multiple routines/techniques, and also has options to include preconditioning. We cannot possibly compare against all options, nor use preconditioners. Therefore, we only provide a small sample and try to only use default values. We only set the parameters needed for comparison. For all examples, we set the parameters to indicate that the problems are real and to use double precision. Also, the value *primme.method* is set to be the *default_min_matvecs*, since this is the measure we are comparing, and finally, the method is set to be *primme_svds_hybrid*. These choices provided very consistent results throughout all examples.

Since the goal of the paper is to use as little storage as possible and if a routine cannot be used with the default values and the fixed parameter for maximum basis size, we recorded the result as "N/A" - not available for that basis size. We did not try to modify parameters to get the routine to work for that basis size. Likewise, if a routine did not converge we recorded the result as "N/C" and also did not modify the

parameters to get the routine to work (e.g., increase the default setting for maximum number of iterations).

The code `trrsvds` implements the following reorthogonalization strategies: applies one-sided full reorthogonalization when matrix A is fairly well conditioned and two-sided full reorthogonalization when A is ill-conditioned. The condition number of A is estimated with minimum and maximum singular values of the computed B_m over all iterations thus far. Examples presented in this section with `trrsvds`, one- and two-sided full reorthogonalization yield about the same accuracy, and so we do not report both. It should be noted that the full reorthogonalization strategy increases the overall computational times when Lanczos basis is increased. Unlike `trrsvds`, reorthogonalization is not used in `rd2svds` since only one-step of the GKL process is used to build 2-GKL factorization. The routines `rd2svds` and `trrsvds` with basis size of only two vectors ($m = 2$) using hybrid method with normal equations and searching only for the largest singular triplets are mathematically equivalent, but they are slightly numerically different (see the results as reported in the examples when $k = 1$ and $m = 2$).

To illustrate the different methods available for `trrsvds` via the parameter choices we used the notation `trrsvds([nor,aug,thk],[ritz,harm])` in all of our examples. The first entry is either `nor` for the normal equations in the hybrid method to compute the iterative refined Ritz pairs (108), `aug` for the augmented equations in the hybrid method to compute the iterative refined Ritz pairs (129) and `thk` for using only the thick–restarted method (non-hybrid), see Algorithm 6.3. The second entry is either `ritz` using Ritz vectors with thick–restarted or `harm` using harmonic Ritz vectors with thick–restarted. For example, `trrsvds(nor,ritz)` uses the normal equations in the hybrid method to compute the iterative refined Ritz pairs and Ritz vectors when using thick–restarted.

The numerical examples are separated into four distinct parts. Example 3.5.1 displays the performance of `trrsvds` and `rd2svds` for computing the largest singular triplets on a diagonal matrix with varying k and m values. The example provides useful illustrations and comparison of methods developed in the paper. Example 3.5.2 is a comparison of the codes on a variety of common test matrices from the SuiteSparse Matrix Collection [10], see Table 5. The example is focused on computing the largest singular triplets for varying k and m values. Example 3.5.3 is a comparison of the methods for finding largest singular triplet of one of the largest test matrices in the SuiteSparse Matrix Collection. Finally, Example 3.5.4 is focused on computing the smallest singular triplets on two commonly used test matrices from the SuiteSparse Matrix Collection.

Table 5. Test matrices used for the examples from the SuiteSparse Matrix Collection [10]

Matrix	Size	Non-zeros	Kind
illc1033	1033×320	4719	Least Squares
lp_ganges	1309×1706	6937	Linear Programming
dwt_1242	1242×1242	10426	Structural
well1850	1850×712	8755	Least Squares
pde2961	2961×2961	14585	2D/3D Problem
Kemelmacher	28452×9693	100875	Graphics/Vision
JP	87616×67320	13734559	Tomography
amazon0302	262111×262111	1234877	Directed Graph
Rucci1	1977885×109900	7791168	Least Squares
relat9	12360060×549336	38955420	Combinatorial
kmerV1r	$214005017 \times 214005017$	465410904	Undir. Graph

In all examples and for all codes except `svdifp`, the matrix A and A^T are only accessed by call to a function with input x and output Ax or A^Tx with an input parameter indicating which product to perform. `svdifp` requires user to input the matrix A. The recorded value mvp in the examples is the total number of times Ax and A^Tx are computed. To aid in computational times A^Tx is performed as

$(x^T A)^T$ to avoid constantly transposing a very large matrix. We also implemented this strategy in `svdifp`. When the matrix C (77) is used, to save memory space, it is never explicitly formed; the input vector is split and the calculation is only performed on Ax and $A^T x$.

The recorded cpu times displayed in the examples are in seconds and recorded using MATLAB's tic-toc command. Since `primme_svds` is a MATLAB interfaced code, the recorded times are expected to be less than the MATLAB codes. We finally remark that the performance of the methods in our comparisons also depends on the machine architecture, the author's coding style, the design/purpose of the routines, and numerical implementation. Our MATLAB codes are only an illustration of the presented methods and the comparison are only meant to show the methods in this paper are competitive to other existing routines.

All numerical examples were carried out using MATLAB version R2021a on a MacBook Pro 2.6 GHz 6-Core Intel Core i7 processor and 16 GB (2667 MHz) of memory using operating system macOS Big Sur. Machine epsilon is $\epsilon = 2.2 \cdot 10^{-16}$. For all examples we set the following: a common number of desired singular values, a common random starting vector, comparable parameters to represent the search space, and $tol = 10^{-6}$ (10^{-10} tolerance is used only in Example 3.5.4). Finally, in all of the figures, "N/A" is used to denote that the method is not available for the specified choice of parameters, "N/C" denotes the routine did not converge in the allotted (default) number of iterations, and, unless otherwise stated, the number above the converged methods denotes the total cpu time in seconds obtained from the MATLAB's tic-toc command.

Example 3.5.1 (Largest Singular Triplets):

Let $A = \text{diag}(1{:}500)$ be a 500×500 diagonal matrix.

3.5.1.a. We are searching for the the largest singular triplets for varying param-

eter choices in the routine `trrsvds`. We let $k = 1, 2, 3, 4$ and vary m from $(k+1)$ to $(k+4)$. For comparison, we also included the similar hybrid MATLAB eigenvalue code `trreigs`[6] [2] applied to the equivalent eigenvalue problem for matrix $C = \left[\begin{smallmatrix} 0 & A \\ A^T & 0 \end{smallmatrix}\right]$. The matrix-vector products of C are recorded as the sum of the matrix-vector products of A and A^T. The code `trreigs` applies the hybrid restarted method using either the iterative refined Ritz vectors or thick–restarted with Ritz vectors to compute the extreme eigenpairs. The starting vector for `trreigs` was the same as `trrsvds` for the last 500 entries and zero values for the first 500 entries.

In Figure 5 we display our results. The recorded values for `rd2svds`, the matrix-vector products, and the number of restarts with iterative refined Ritz vectors are the total values necessary to get all of the desired values within the tolerance. Also, recall from Remark 3.5 that `rd2svds` is only reported for size $m = k + 1$. The values above the bars in Figure 5 correspond to the number of restarts with iterative refined Ritz vectors. As expected, from Figure 5 we observe that for small m and small k values many restarts are needed with iterative refined Ritz vectors. As m and/or the number of desired singular triplets k increase, the routine uses less restarts with iterative refined Ritz vectors. This can contributed in part to the criteria (141) needed to be satisfied becomes more difficult. However, even a few restarts with iterative refined Ritz vectors can still improve overall convergence, particularly when m is small. The method `trreigs` applied to C did not perform as well for all m values and $k > 1$ in comparison with `trrsvds`. This can be contributed in part to the larger spread of eigenvalues of C. Therefore, we excluded `trreigs` from the other examples and in turn used `eigs(C)` for comparisons. We have also excluded `trrsvds(thk,--)` from the other examples given its similarities with `irlba`.

In summary, this example is just one illustration that the hybrid method `trrsvds` is better suited for problems when searching for small number of singular triplets k

[6]Code available at: `http://www.math.uri.edu/~jbaglama`

given small m.

3.5.1.b. We continue with the same diagonal matrix and compare five methods from the paper, namely `rd2svds`, `trrsvds(nor,ritz)`, `trrsvds(aug,ritz)`, `trrsvds(nor,harm)`, and `trrsvds(aug,harm)`, with the methods `eigs(C)`, `irlba`, `svdifp`, `svds`, and `primme_svds`. Figure 6, which displays the result of this comparison, shows that all five methods from the paper converged for all m and k values, particularly of note is the case when $m = k+1$. This example shows that the methods developed in the paper are particularly competitive when using small m relative to the desired number of singular triplets k. As m and k values increase we see the routines in this paper remain fairly consistent and competitive among the other publicly available codes. This pattern continues for the rest of the examples.

Example 3.5.2 (Largest Singular Triplets):

For this example, we are using nine test matrices listed in Table 5, namely illc1033, dwt_1242, well1850, pde2961, Kemelmacher, JP, amazon0302, Rucci1, and relat9. We are searching for the the largest singular triplets and let $k = 1, 2, 3, 4$ and vary m from $(k + 1)$ to $(k + 4)$. As in the previous example, we consider ten methods and display the results comparing matrix-vector products and cpu times for all routines in Figures 7-15. The four largest singular values and the order of the maximum residual errors for the routines can be found in captions of the respective figures. We see that all five methods from the paper compared very favorably against other routines for all m and k values. Moreover, the residual norms are all of the same order of magnitude. Given a wide range in sizes of the test matrices, together with the varied proximity among the largest singular values, these examples reinforce our previous observation — methods developed here are particularly competitive when using very small m relative to the desired number of singular triplets.

Example 3.5.3 (Largest Singular Triplet):

Table 6. Example 3.5.3. Computing the largest singular triplet for matrix kmerV1r with $m = 2, 3$. The residual norm $\sqrt{\|Av_1 - \sigma_1 u_1\|^2 + \|A^T u_1 - \sigma_1 v_1\|^2}$ was computed explicitly from the output from each routine.

Method	m	mvp A and A^T	mvp cpu	total cpu	residual norm
`rd2svds`	2	72	1770s	2478s	$1.5 \cdot 10^{-6}$
	-	-	-	-	-
`trrsvds(nor,ritz)`	2	66	1652s	2598s	$5.3 \cdot 10^{-6}$
	3	82	1846s	3383s	$1.7 \cdot 10^{-6}$
`trrsvds(aug,ritz)`	2	80	1894s	3066s	$6.3 \cdot 10^{-6}$
	3	66	1467s	2645s	$5.6 \cdot 10^{-6}$
`trrsvds(nor,harm)`	2	74	1774s	2842s	$5.5 \cdot 10^{-6}$
	3	74	1654s	3105s	$6.1 \cdot 10^{-6}$
`trrsvds(aug,harm)`	2	92	2148s	3723s	$6.3 \cdot 10^{-6}$
	3	66	1475s	2977s	$3.4 \cdot 10^{-6}$
`eigs(C)`	N/A	-	-	-	-
	3	274	8995s	25712s	$8.7 \cdot 10^{-6}$
`irlba`	2	138	3124s	5473s	$6.0 \cdot 10^{-6}$
	3	90	1941s	3608s	$3.5 \cdot 10^{-6}$
`svdifp`	2	81	-	8049s	$5.7 \cdot 10^{-6}$
	3	75	-	8457	$2.3 \cdot 10^{-6}$
`svds`	N/A	-	-	-	-
	3	206	6407s	15232s	$5.1 \cdot 10^{-6}$
`primme_svds`	N/A	-	-	-	-
	3	64	1413s	2438s	$3.6 \cdot 10^{-6}$

For this example, we are computing the largest singular triplet for the matrix kmerV1r, currently the second largest in order in the SuiteSparse Matrix Collection. This is also one of the largest matrices that was able to be loaded into MATLAB allowing all of the routines to successfully compute the largest singular triplet. This example pushed the bounds of the machine architecture used. Table 6 displays the results for computing the largest singular triplet for $m = 2, 3$. The largest singular value was computed by all routines as $\sigma_1 = 6.5035$. As seen in Table 6, for $m = 2$ our MATLAB codes `ttrsvds` and `rd2svds` all converged in approximately one hour total time, the fastest converging in about 41 minutes. This is comparable to the MATLAB interfaced code `primme_svds` with $m = 3$ (not available at $m = 2$). MATLAB's

internal functions `eigs(C)` and `svds` were only available for $m = 3$ and needed more than 7 hours and 4 hours, respectively.

Example 3.5.4 (Smallest Singular Triplets):
Although the focus of the developed methods is not on computing the smallest singular triplets, we still include two examples. We consider two commonly used test matrices in the literature, lp_ganges and well1850. For this example, we let $k = 1, 2, 3, 4$ and vary the maximum basis size from 12 to 15 – it is well-known that when computing the smallest singular triplets a larger basis size is required, see e.g., [5, 25].

We start with some remarks. Our routine `rd2svds` could not be used, since it requires the basis to be fixed at 2. The routine `eigs(C)` could not be used with only matrix-vector products since the smallest singular triplets are interior values. `eigs` also requires a linear solver routine, i.e., $(A - \sigma I)\backslash x$. The routine `svds` also could not be used with only matrix-vector products. Moreover, `svds` requires the QR decomposition of the input matrix A and then searches for the largest singular triplets using linear solve routines $R\backslash x$ and $R^T\backslash x$. Therefore, in Figures 16-17, N/A is used to report that these three methods are not available.

For the remaining seven methods, we set $tol = 10^{-10}$ and display the results comparing matrix-vector products and cpu times in Figures 16-17. The four smallest singular values along with the maximum residual errors can be found in the captions of Figures 16-17. As we can see, all of the four available methods from this paper compared favorably against the other routines for all m and k values. It should be noted again that the routines `svdifp` and `primme_svds` allow use of preconditioners which have been shown to significantly improve their performance in comparison when no preconditioner is used, see [33, 53]. Therefore, our presented results are not a reflection on the performance of those codes as developed. However, as previously stated, preconditioners (and factorization - QR) increase storage requirements, and

therefore in this context are not used.

In summary, even though our developed methods focused on the computation of the largest singular triplets, they proved to be competitive in case when smallest singular triplets are desired.

3.6 Conclusions

This paper extends the hybrid concept in [2] recently applied to the symmetric eigenvalue problem to the GKL process. The new restarted hybrid GKL method combines thick–restarting with Ritz or harmonic Ritz vectors or with a judiciously chosen linear combination of iterative refined Ritz vectors. Numerical examples show the method to be highly competitive with other publicly available codes, particularly when there are limited memory requirements.

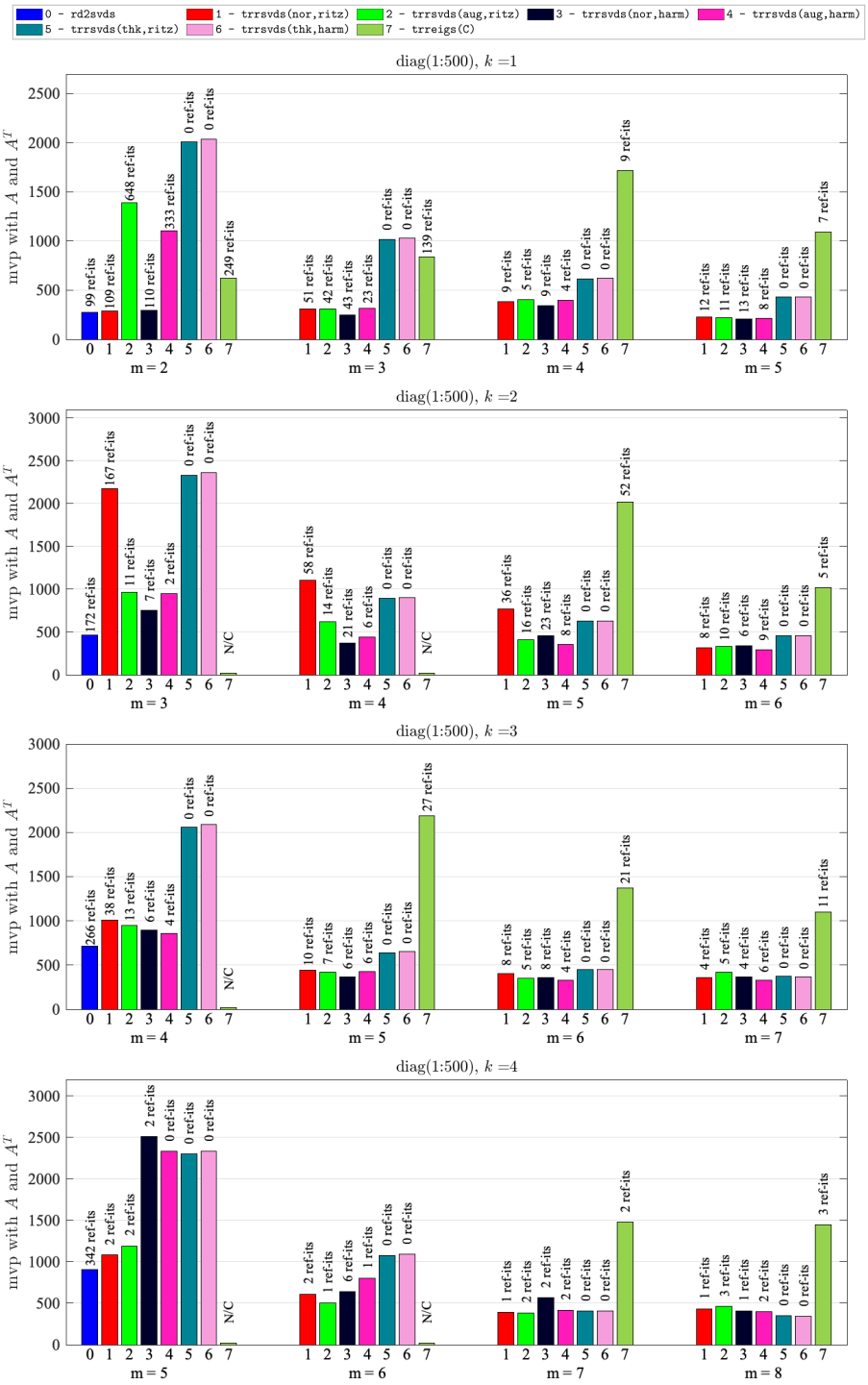


Figure 5. Example 3.5.1.a. (largest singular triplets) $A = \text{diag}(1{:}500)$ varying values of k and basis size for MATLAB codes `rd2svds`, `trrsvds` and `trreigs(C)`. The value above the bars are the number of restarts with iterative refined Ritz vectors. For small m and small k values more restarts are required with iterative refined Ritz vectors.

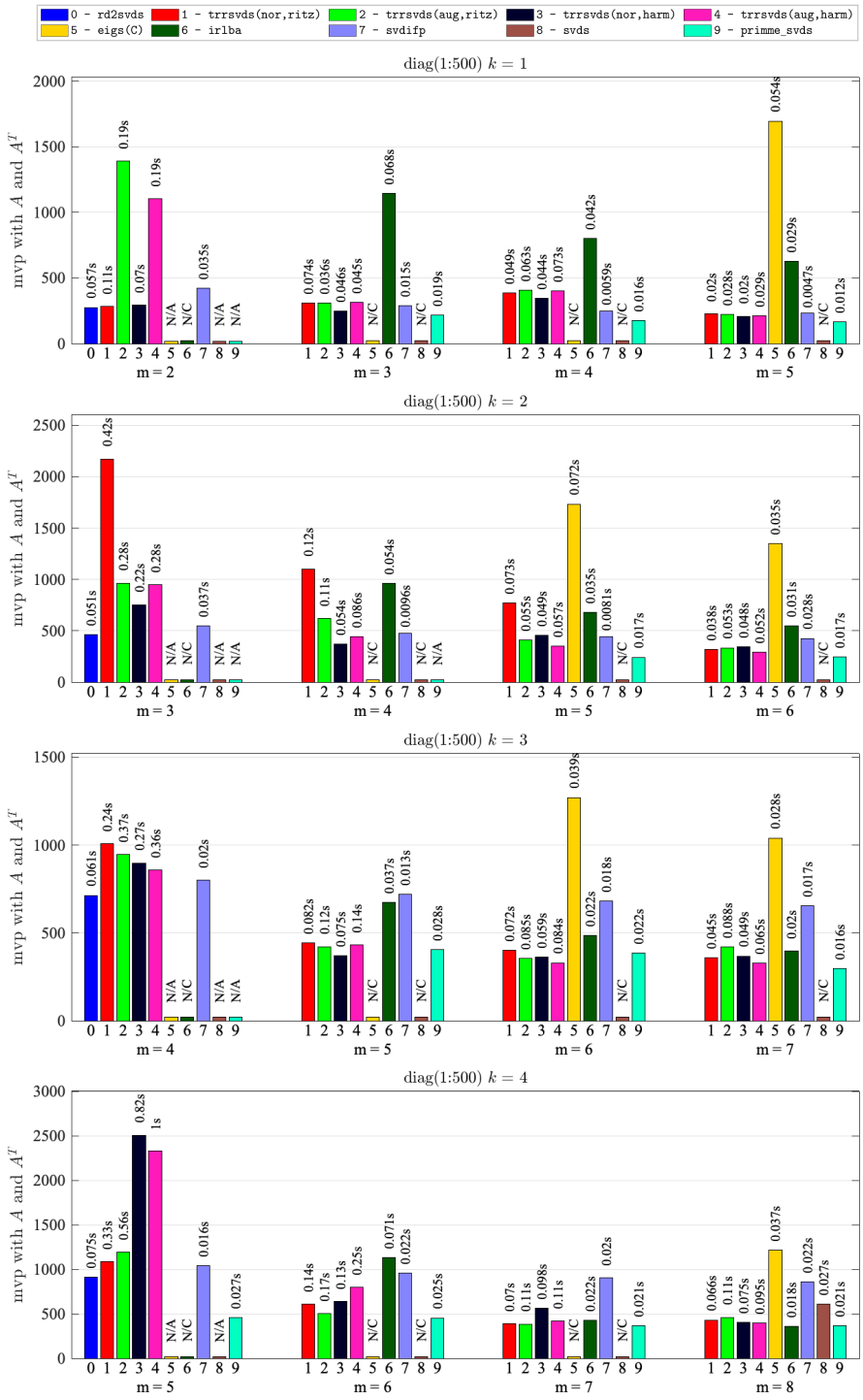


Figure 6. Example 3.5.1.b. (largest singular triplets) $A = \text{diag}(1{:}500)$ varying values of k and m for different MATLAB codes.

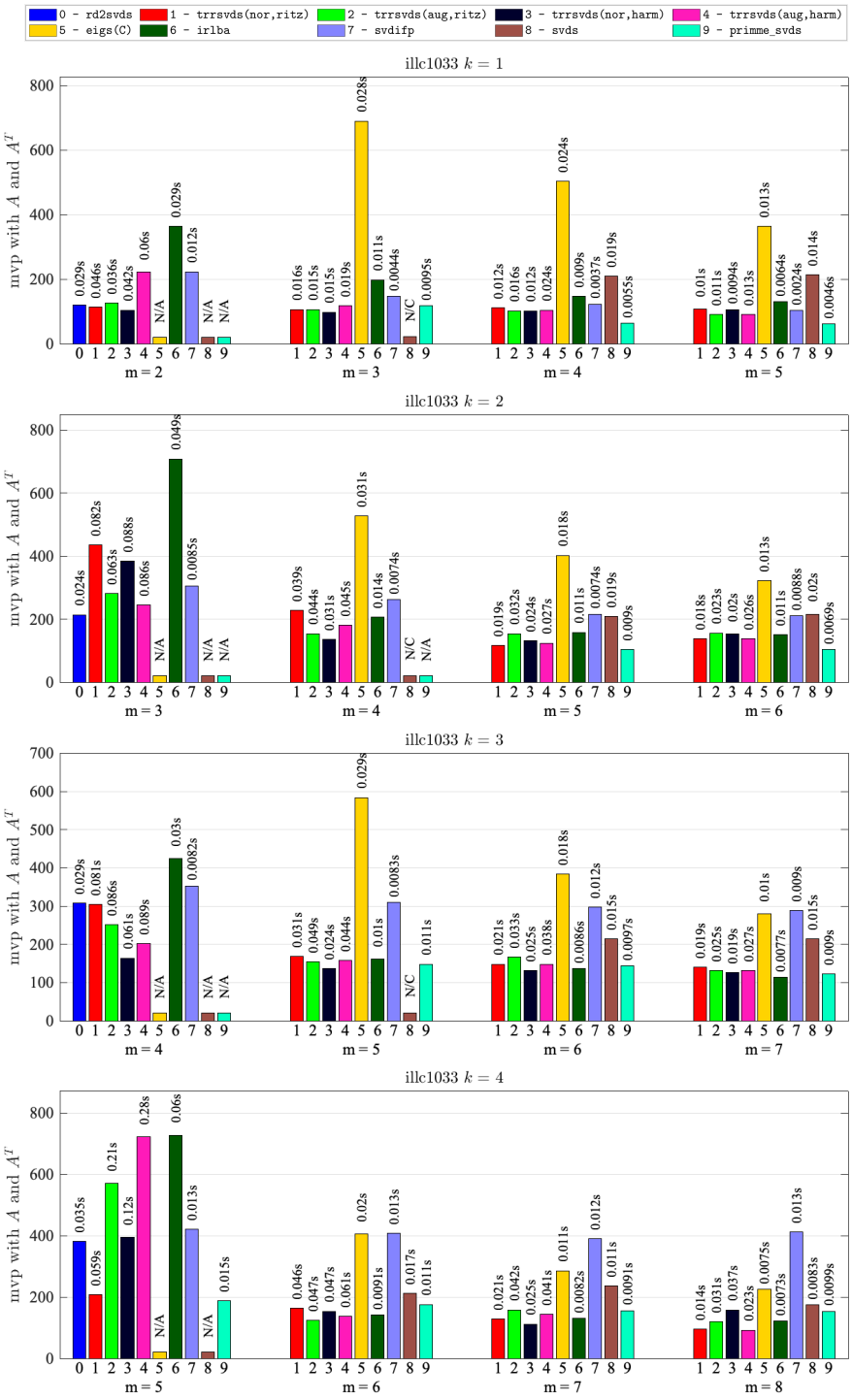


Figure 7. Example 3.5.2 (largest singular triplets) for the 1033×320 matrix illc1033 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-6}. The four largest singular values are: $\sigma_1 = 2.1444$, $\sigma_2 = 2.1042$, $\sigma_3 = 2.0885$, and $\sigma_4 = 2.0574$.

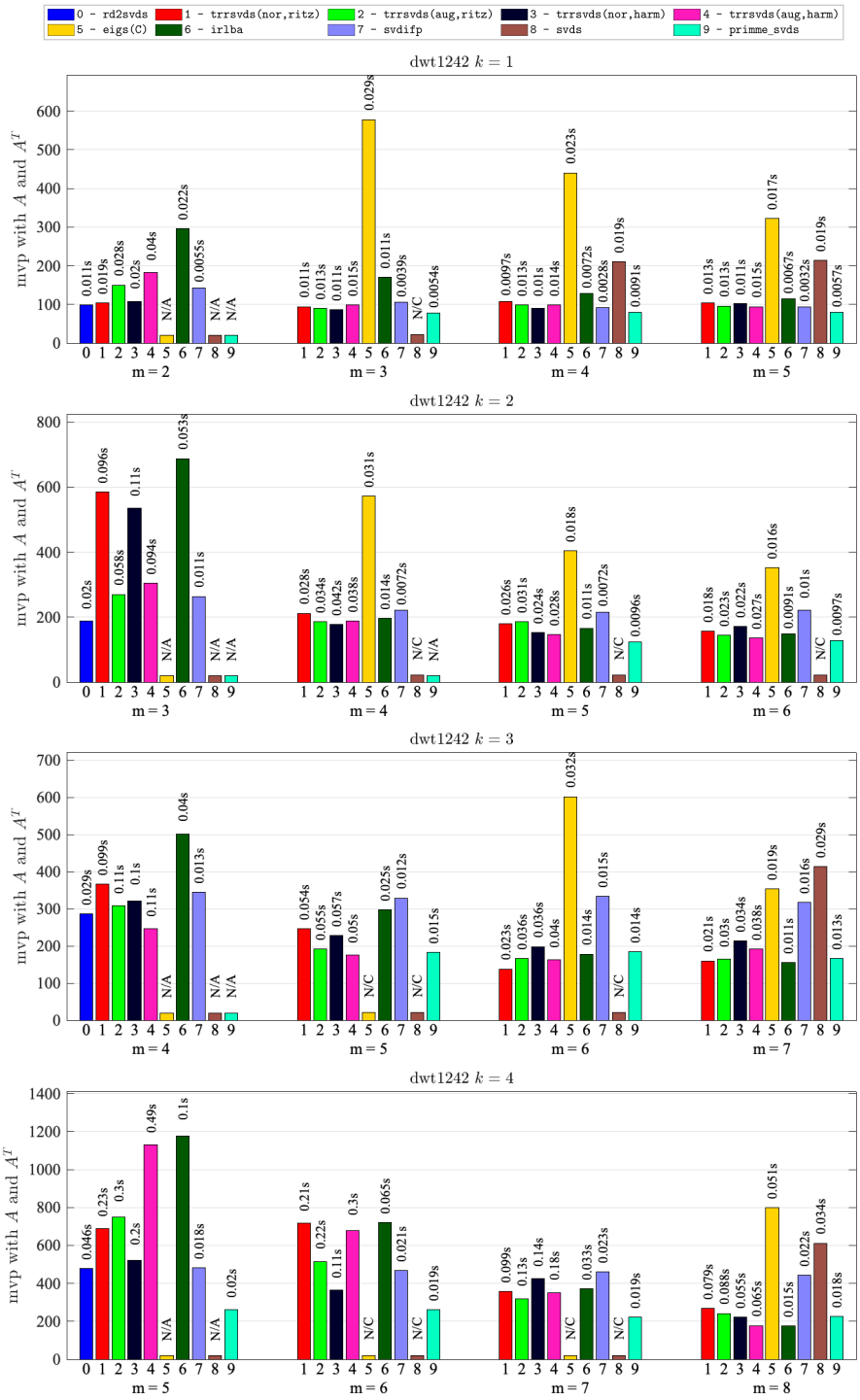


Figure 8. Example 3.5.2 (largest singular triplets) for the 1242×1242 matrix dwt_1242 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-6}, except for `eigs(C)` it is $\approx 1.3 \cdot 10^{-5}$. The four largest singular values are: $\sigma_1 = 9.3912$, $\sigma_2 = 9.2379$, $\sigma_3 = 9.1552$, and $\sigma_4 = 9.0722$.

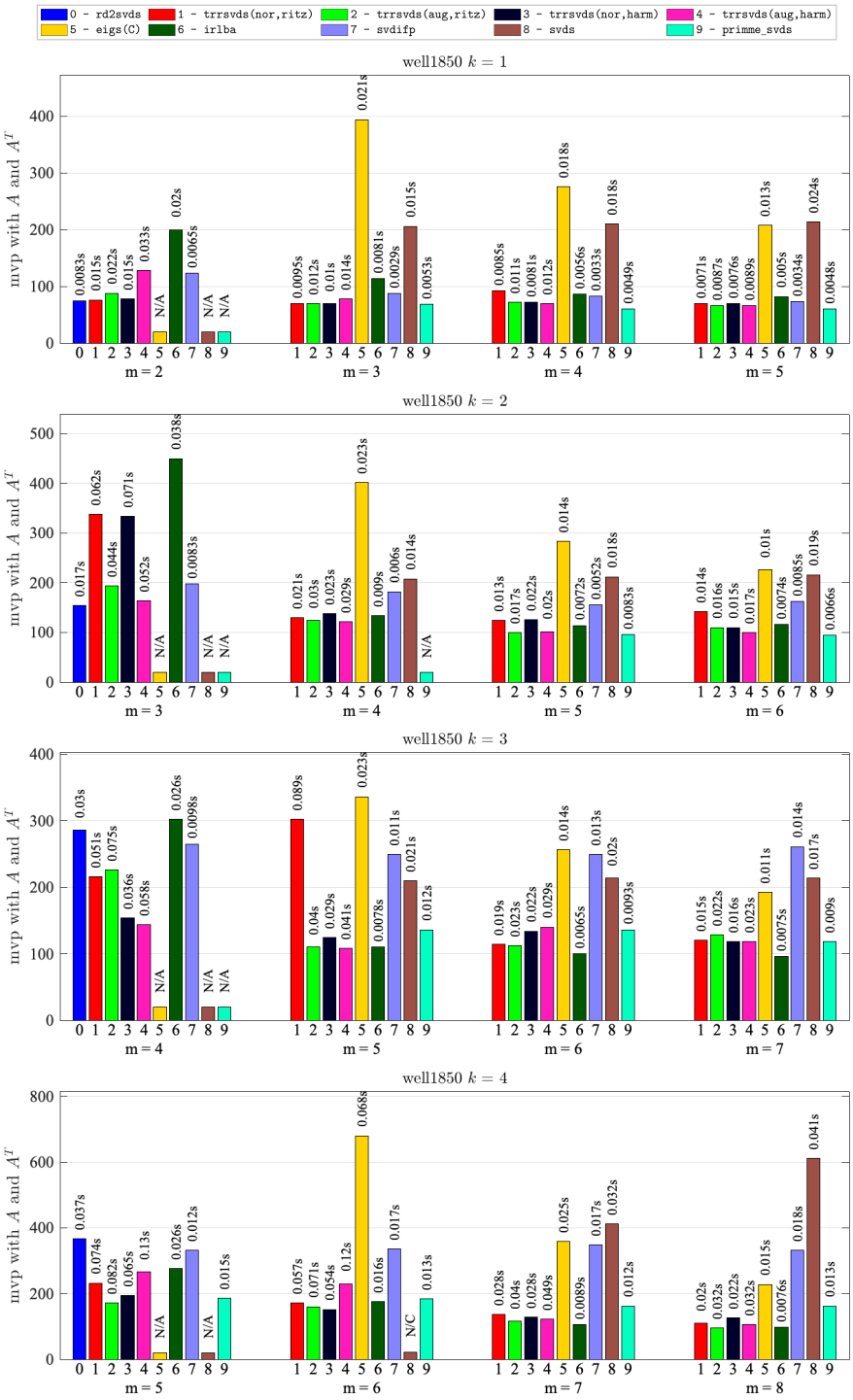


Figure 9. Example 3.5.2 (largest singular triplets) for the 1850×712 matrix well1850 for varying values of k and maximum (basis) size for different routines. For all methods, the maximum residual errors are on the order of 10^{-6}. The four largest singular values are: $\sigma_1 = 1.7943$, $\sigma_2 = 1.7388$, $\sigma_3 = 1.7189$, and $\sigma_4 = 1.6828$.

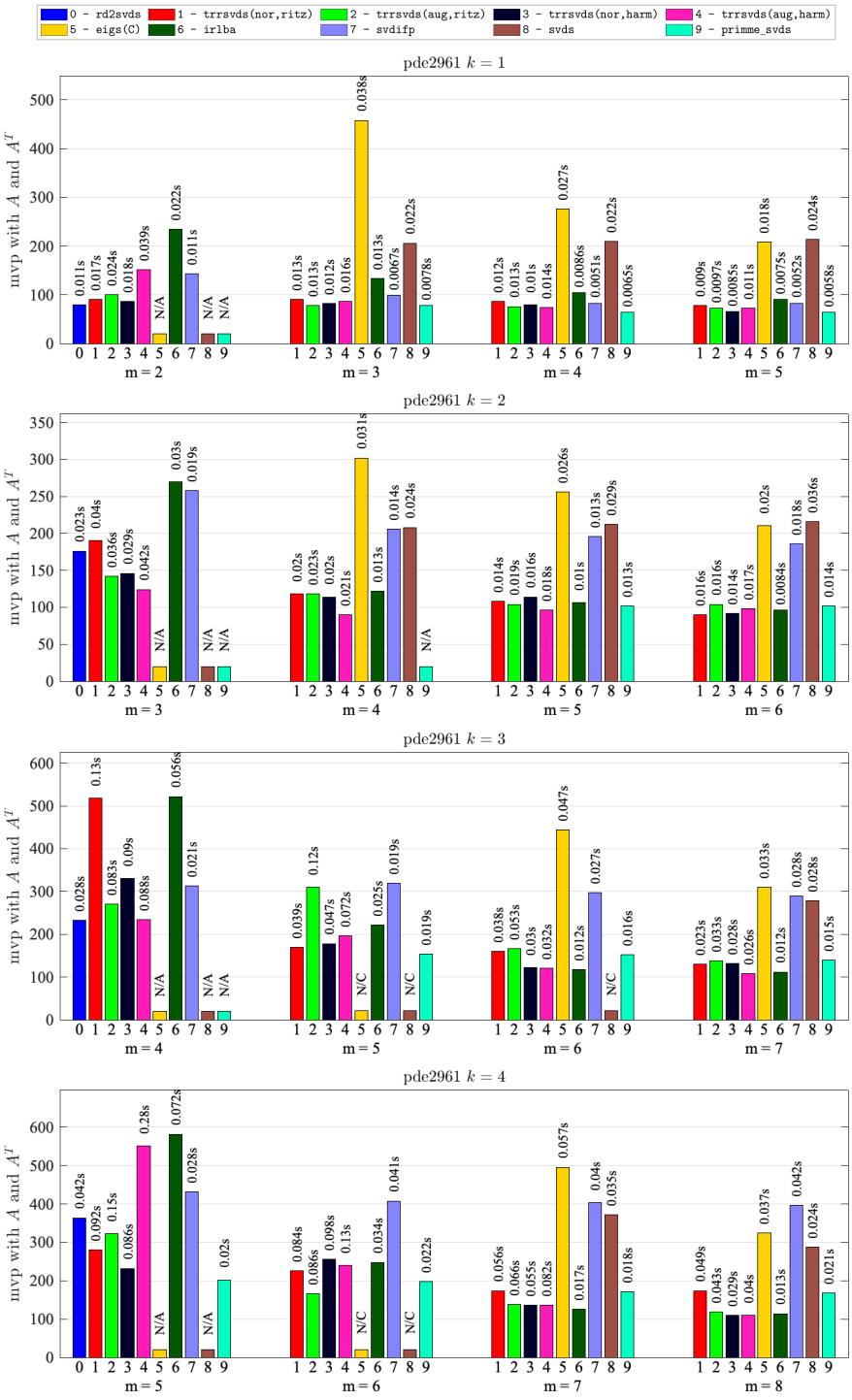


Figure 10. Example 3.5.2 (largest singular triplets) for the 2961×2961 matrix pde2961 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-5}, except for `rd2svds` and `svds` they were $\approx 9.8 \cdot 10^{-6}$. The four largest singular values are: $\sigma_1 = 10.378$, $\sigma_2 = 10.096$, $\sigma_3 = 9.8779$, and $\sigma_4 = 9.7856$.

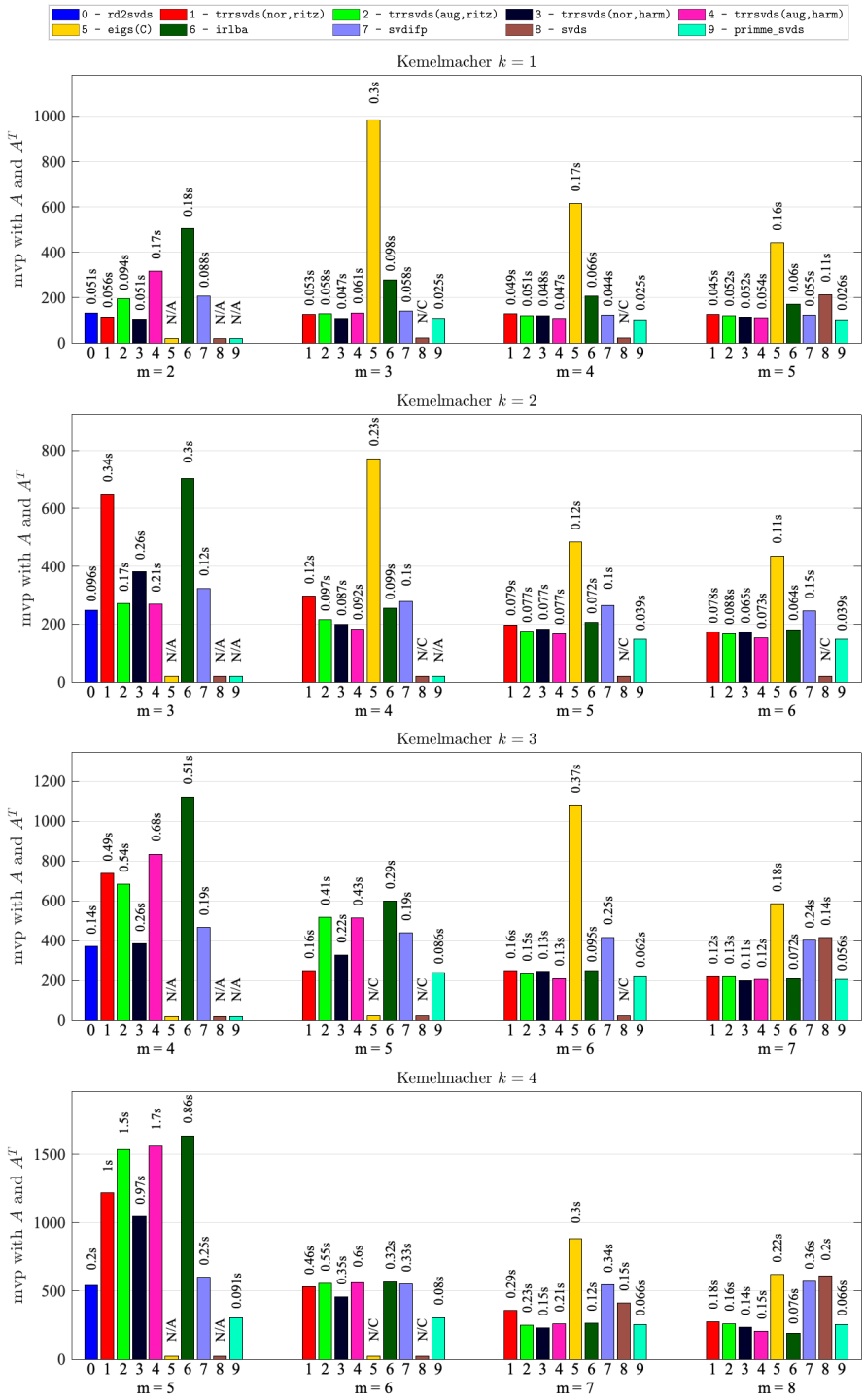


Figure 11. Example 3.5.2 (largest singular triplets) for the 28452×9693 matrix Kemelmacher for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-4}. The four largest singular values are: $\sigma_1 = 240.58$, $\sigma_2 = 238.02$, $\sigma_3 = 236.17$, and $\sigma_4 = 235.35$.

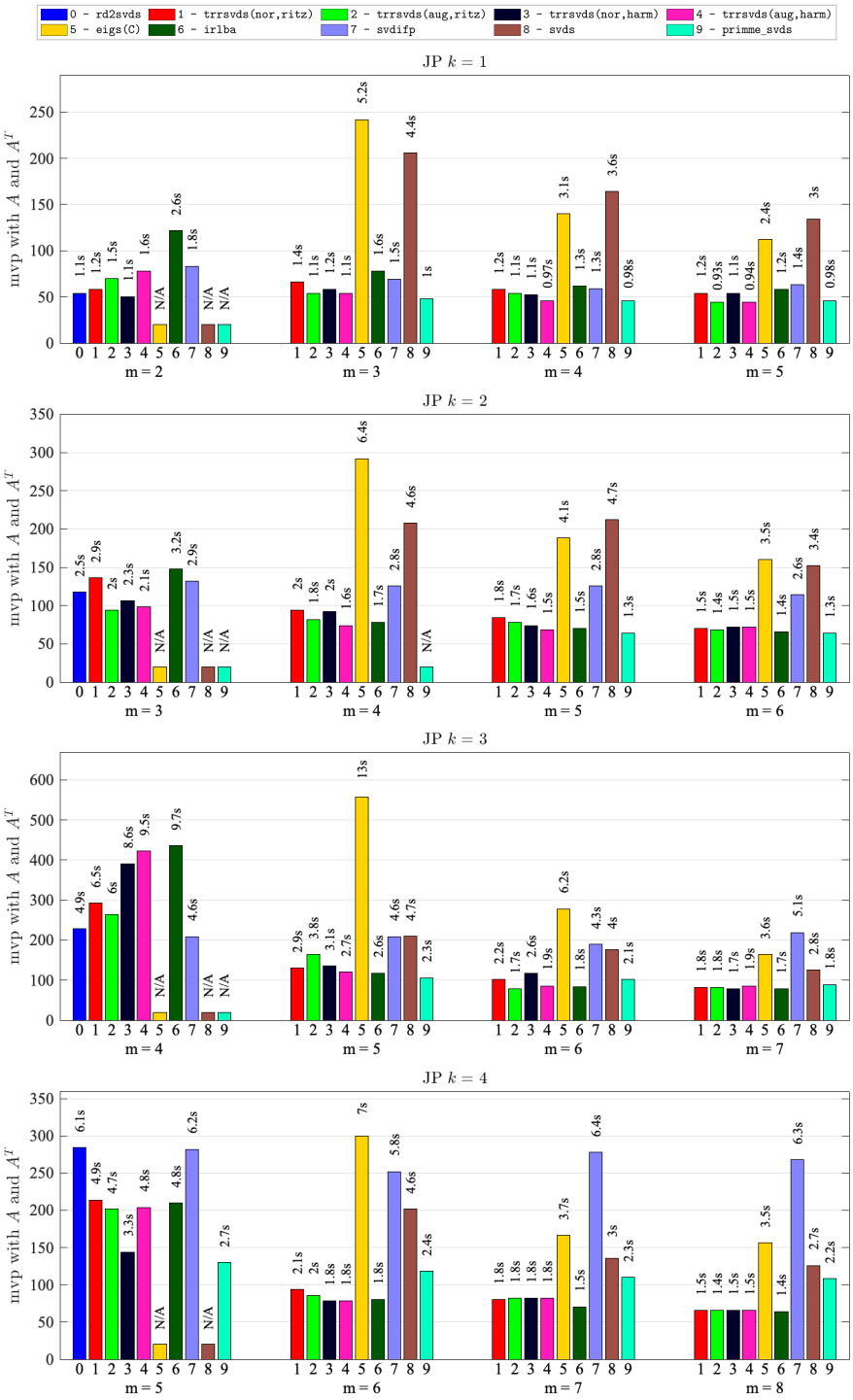


Figure 12. Example 3.5.2 (largest singular triplets) for the 87616×67320 matrix JP for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-3}. The four largest singular values are: $\sigma_1 = 4223.1$, $\sigma_2 = 4019.3$, $\sigma_3 = 3872.8$, and $\sigma_4 = 3819.2$.

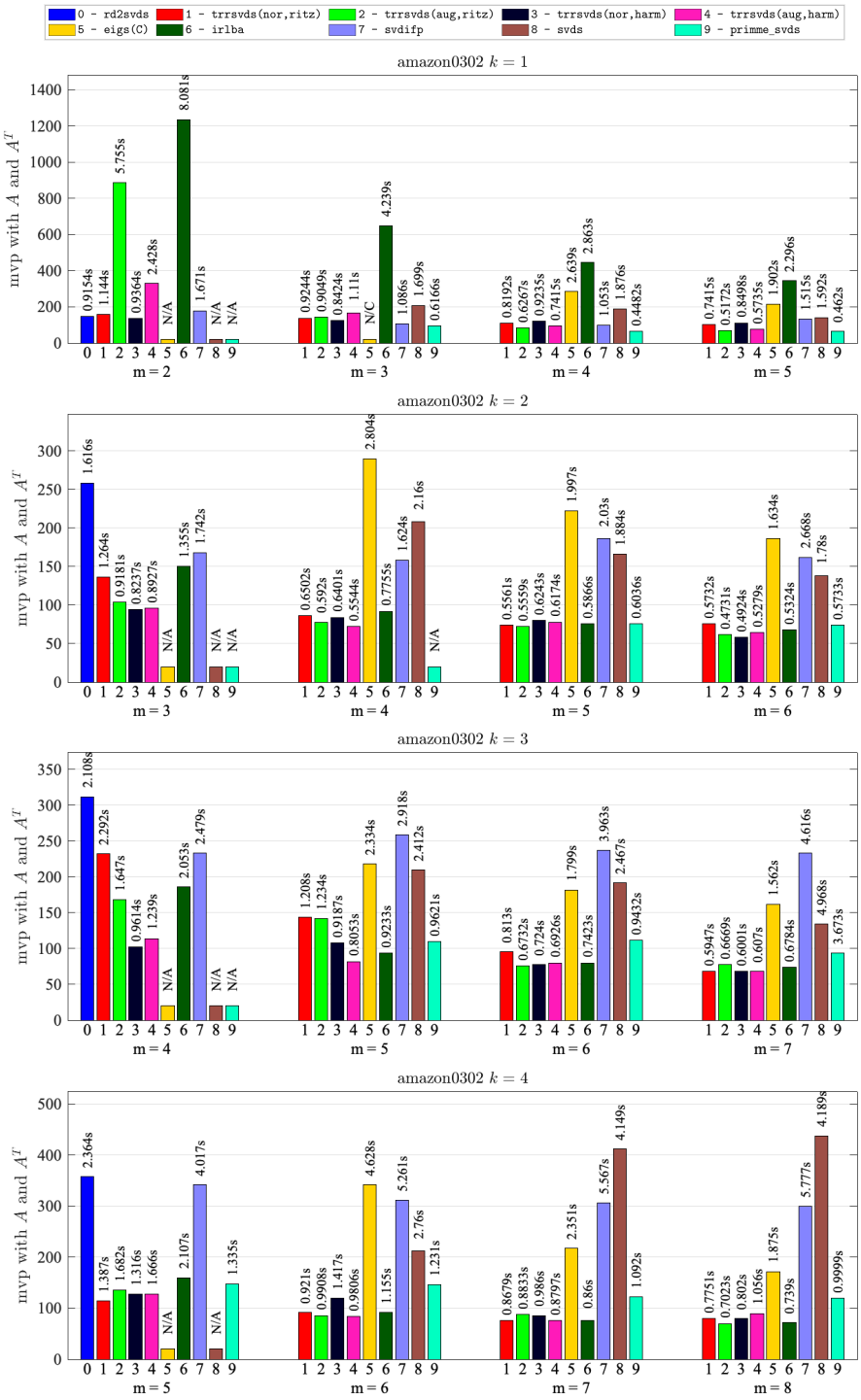


Figure 13. Example 3.5.2 (largest singular triplets) for the 262111×262111 matrix amazon0302 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-5}. The four largest singular values are: $\sigma_1 = 21.218$, $\sigma_2 = 21.136$, $\sigma_3 = 20.027$, and $\sigma_4 = 19.277$.

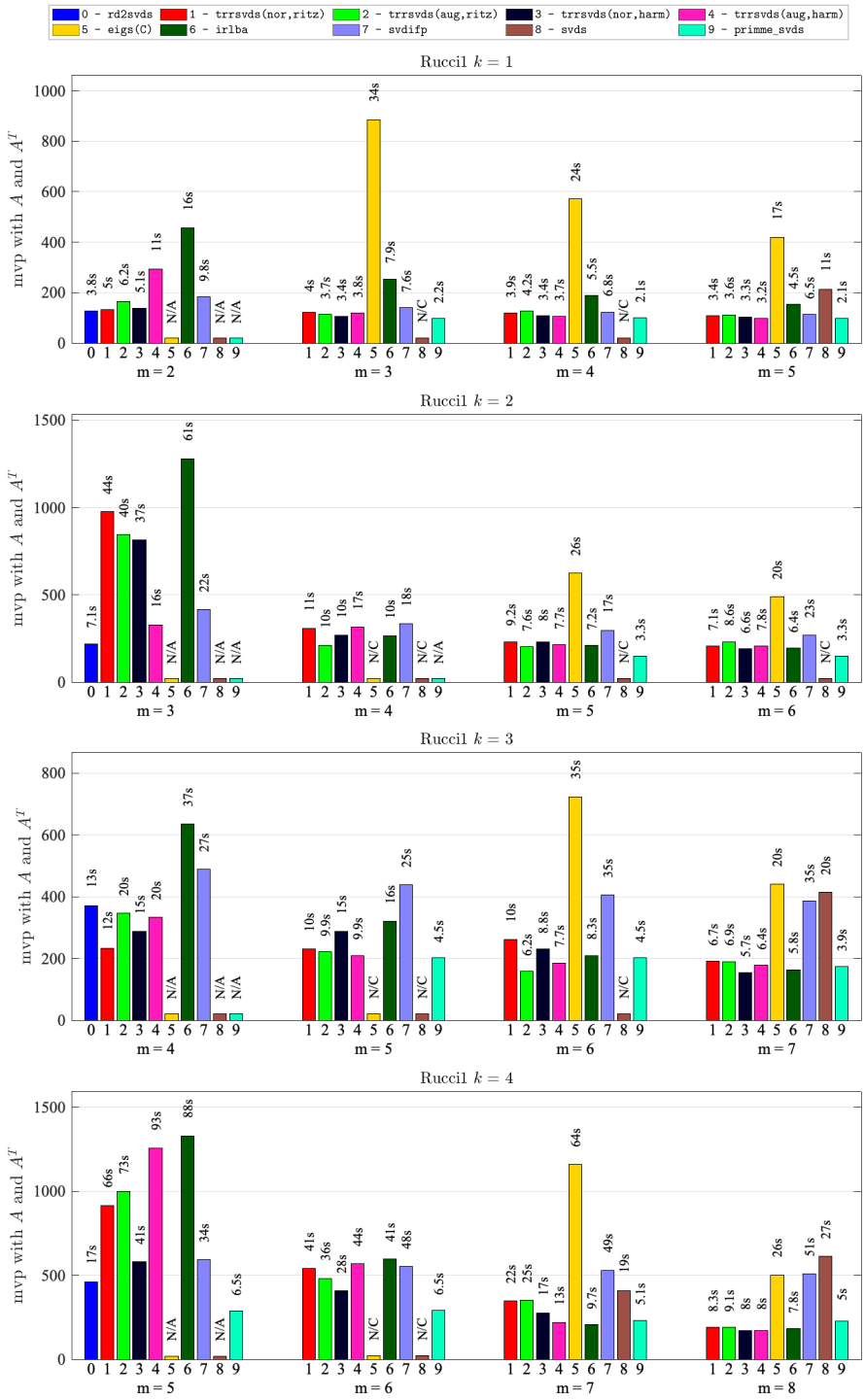


Figure 14. Example 3.5.2 (largest singular triplets) for the 1977885×109900 matrix Rucci1 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-6}. The four largest singular values are: $\sigma_1 = 7.0687$, $\sigma_2 = 6.9853$, $\sigma_3 = 6.9625$, and $\sigma_4 = 6.8895$.

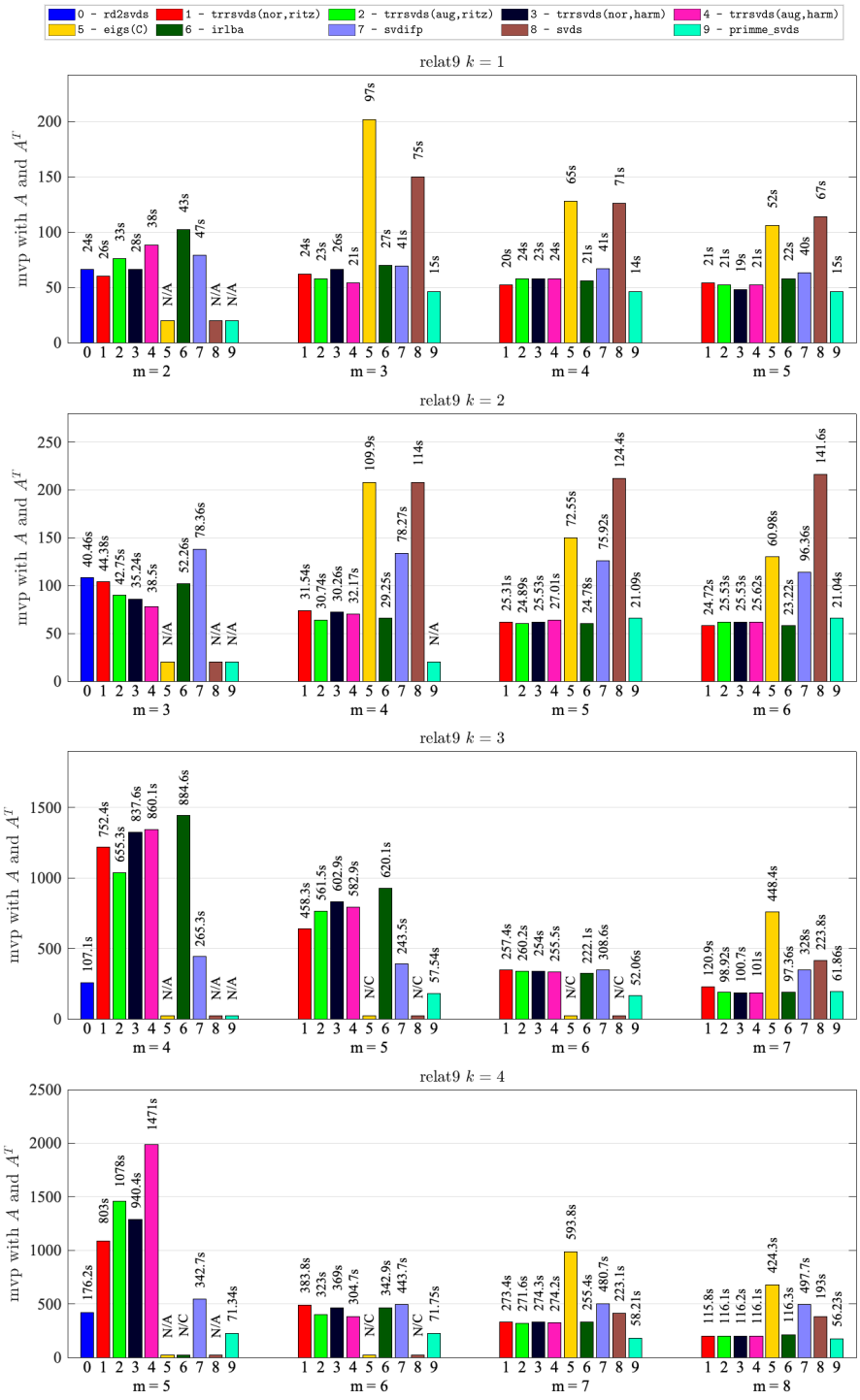


Figure 15. Example 3.5.2 (largest singular triplets) for the 12360060×549336 matrix relat9 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-5}. The four largest singular values are: $\sigma_1 = 21.626$, $\sigma_2 = 20.417$, $\sigma_3 = 18.666$, and $\sigma_4 = 18.61$.

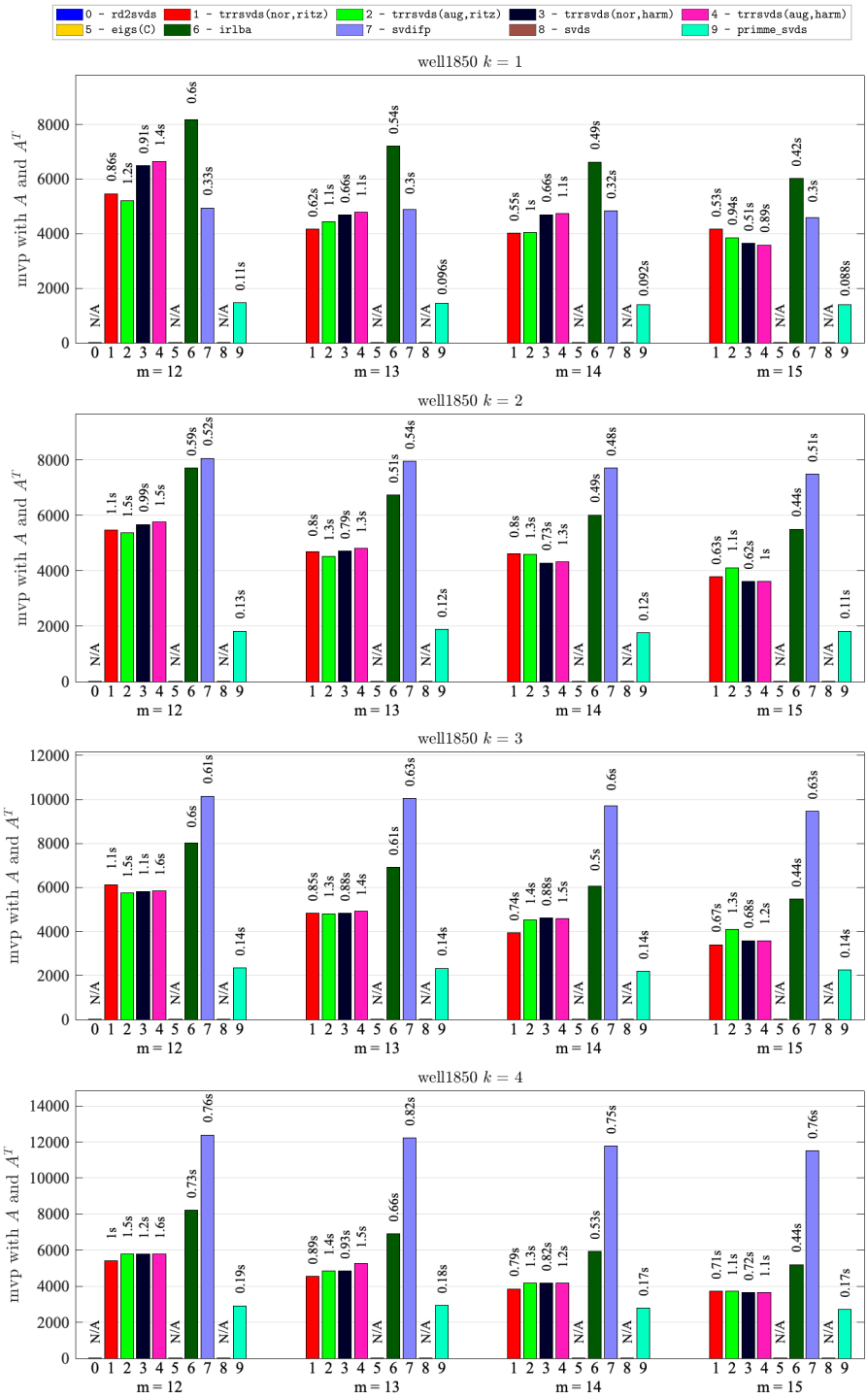


Figure 16. Example 3.5.4 (smallest singular values) for the 1850×712 matrix well1850 for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-10}. The four smallest singular values are: $\sigma_1 = 0.01612$, $\sigma_2 = 0.019113$, $\sigma_3 = 0.02316$, and $\sigma_4 = 0.030219$.

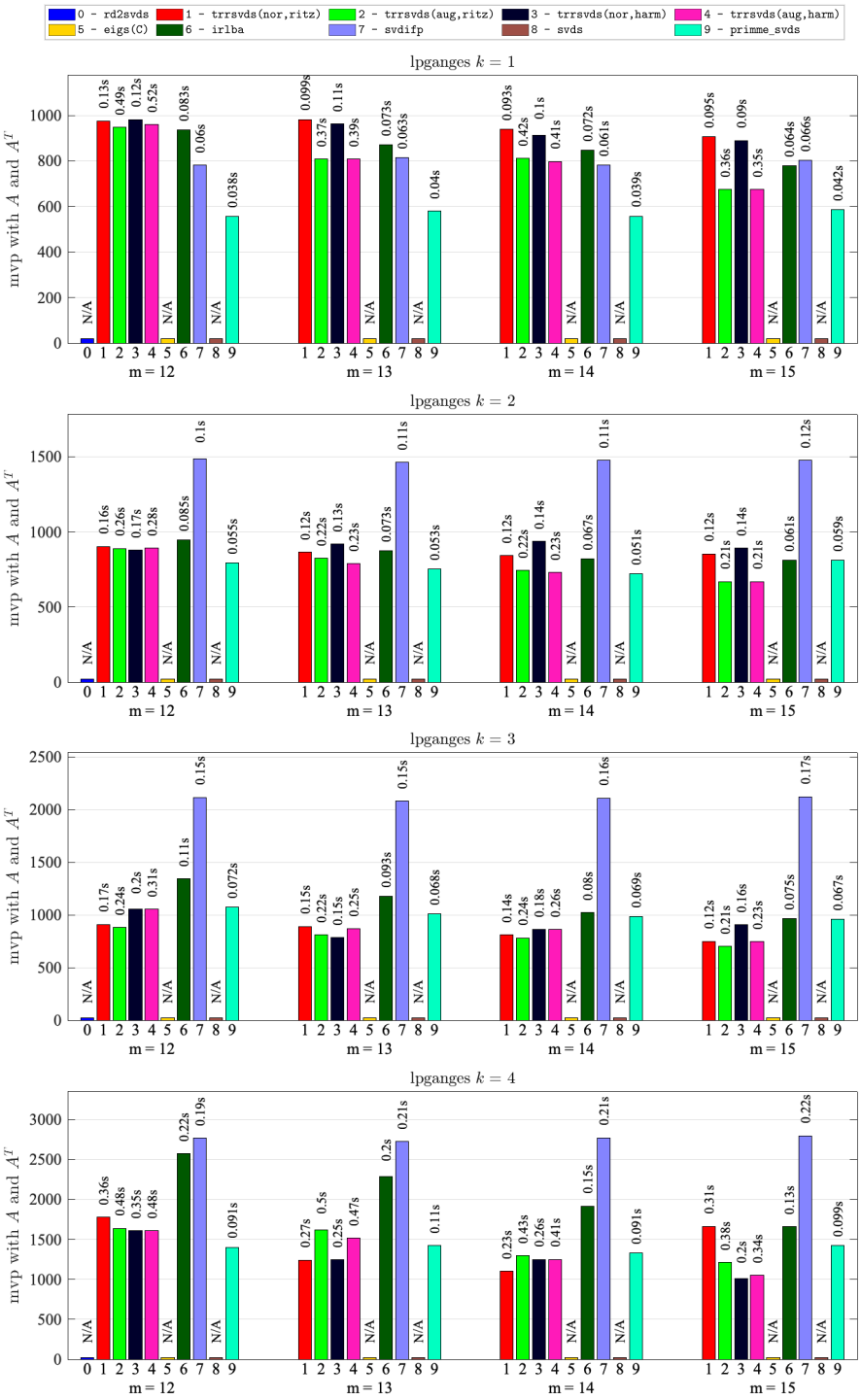


Figure 17. Example 3.5.4 (smallest singular values) for the 1309×1706 matrix lp_ganges for varying values of k and m for different routines. For all methods, the maximum residual errors are on the order of 10^{-10}. The four smallest singular values are: $\sigma_1 = 1.8708 \cdot 10^{-4}$, $\sigma_2 = 0.10645$, $\sigma_3 = 0.16297$, and $\sigma_4 = 0.2079$.

List of References

[1] Alter, O., Brown, P.O., Botstein, D.: Singular value decomposition for genome-wide expression data processing and modeling. Proceedings of the National Academy of Sciences **97**(18), 10101–10106 (2000)

[2] Baglama, J., Bella, T., Picucci, J.: Hybrid iterative refined method for computing a few extreme eigenpairs of a symmetric matrix. SIAM Journal on Scientific Computing Special Session on Iterative Methods (2021 in press). DOI 10.1137/20M1344834

[3] Baglama, J., Calvetti, D., Reichel, L.: IRBL: An implicitly restarted block-Lanczos method for large-scale Hermitian eigenproblems. SIAM Journal on Scientific Computing **24**(5), 1650–1677 (2003)

[4] Baglama, J., Kane, M., Lewis, B., Poliakov, A.: Efficient thresholded correlation using truncated singular value decomposition. arXiv preprint arXiv:1512.07246 (2015)

[5] Baglama, J., Reichel, L.: Augmented implicitly restarted Lanczos bidiagonalization methods. SIAM Journal on Scientific Computing **27**(1), 19–42 (2005)

[6] Baglama, J., Reichel, L.: An implicitly restarted block Lanczos bidiagonalization method using Leja shifts. BIT Numerical Mathematics **53**(2), 285–310 (2013)

[7] Baglama, J., Richmond, D.J.: Implicitly restarting the LSQR algorithm. Electronic Transactions on Numerical Analysis **42**, 85–105 (2014)

[8] Beattie, C.: Harmonic ritz and lehmann bounds. Electronic Transactions on Numerical Analysis **7**, 18–39 (1998)

[9] Berry, M.W.: SVDPACK: A Fortran-77 software library for the sparse singular value decomposition. University of Tennessee (1992)

[10] Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software **38**(1), 1–25 (2011)

[11] Eldén, L.: Matrix methods in data mining and pattern recognition. SIAM (2007)

[12] Goldenberg, S., Stathopoulos, A., Romero, E.: A Golub–Kahan Davidson method for accurately computing a few singular triplets of large sparse matrices. SIAM Journal on Scientific Computing **41**(4), A2172–A2192 (2019)

[13] Golub, G., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis **2**(2), 205–224 (1965)

[14] Golub, G.H., Van Loan, C.F.: Matrix Computations, fourth edn. The Johns Hopkins University Press (2013)

[15] Hochstenbach, M.E.: Harmonic and refined extraction methods for the singular value problem, with applications in least squares problems. BIT Numerical Mathematics **44**(4), 721–754 (2004)

[16] Hochstenbach, M.E.: Generalizations of harmonic and refined Rayleigh-Ritz. Electronic Transactions on Numerical Analysis **20**, 235–252 (2005)

[17] Hochstenbach, M.E., Sleijpen, G.L.: Harmonic and refined Rayleigh–Ritz for the polynomial eigenvalue problem. Numerical Linear Algebra with Applications **15**(1), 35–54 (2008)

[18] Jang, J.G., Choi, D., Jung, J., Kang, U.: Zoom-SVD: Fast and memory efficient method for extracting key patterns in an arbitrary time range. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp. 1083–1092 (2018)

[19] Jia, Z.: Refined iterative algorithms based on Arnoldi's process for large unsymmetric eigenproblems. Linear Algebra and its Applications **259**, 1–23 (1997)

[20] Jia, Z.: Polynomial characterizations of the approximate eigenvectors by the refined Arnoldi method and an implicitly restarted refined Arnoldi algorithm. Linear Algebra and its Applications **287**(1-3), 191–214 (1999)

[21] Jia, Z.: The refined harmonic Arnoldi method and an implicitly restarted refined algorithm for computing interior eigenpairs of large matrices. Applied Numerical Mathematics **42**(4), 489–512 (2002)

[22] Jia, Z.: Some theoretical comparisons of refined Ritz vectors and Ritz vectors. Science in China Series A: Mathematics **47**(1), 222–233 (2004)

[23] Jia, Z.: The convergence of harmonic Ritz values, harmonic Ritz vectors and refined harmonic Ritz vectors. Mathematics of Computation **74**(251), 1441–1456 (2005)

[24] Jia, Z., Niu, D.: An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition. SIAM Journal on Matrix Analysis and Applications **25**(1), 246–265 (2003)

[25] Jia, Z., Niu, D.: A refined harmonic Lanczos bidiagonalization method and an implicitly restarted algorithm for computing the smallest singular triplets of large matrices. SIAM Journal on Scientific Computing **32**(2), 714–744 (2010)

[26] Jia, Z., Stewart, G.W.: An analysis of the Rayleigh–Ritz method for approximating eigenspaces. Mathematics of Computation **70**(234), 637–647 (2001)

[27] Jiang, W., Wu, G.: A thick-restarted block Arnoldi algorithm with modified Ritz vectors for large eigenproblems. Computers & Mathematics with Applications **60**(3), 873–889 (2010)

[28] Jolliffe, I.: Principal component analysis. Wiley Online Library (2005)

[29] Kokiopoulou, E., Bekas, C., Gallopoulos, E.: Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization. Applied Numerical Mathematics **49**, 39–61 (2004). DOI 10.1016/j.apnum.2003.11.011

[30] Larsen, R.: Combining implicit restart and partial reorthogonalization in Lanczos bidiagnalization, 2001

[31] Larsen, R.M.: Propack-Software for large and sparse SVD calculations. Available online. URL http://sun. stanford. edu/rmunk/PROPACK pp. 2008–2009 (2004)

[32] Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. SIAM (1998)

[33] Liang, Q., Ye, Q.: Computing singular values of large matrices with an inverse-free preconditioned Krylov subspace method. Electronic Transactions on Numerical Analysis **42**, 197 (2014)

[34] Morgan, R.B.: On restarting the Arnoldi method for large nonsymmetric eigenvalue problems. Mathematics of Computation **65**(215), 1213–1230 (1996)

[35] Morgan, R.B.: Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. SIAM Journal on Matrix Analysis and Applications **21**(4), 1112–1135 (2000)

[36] Morgan, R.B., Zeng, M.: Harmonic projection methods for large non-symmetric eigenvalue problems. Numerical linear algebra with applications **5**(1), 33–55 (1998)

[37] Niu, D., Yuan, X.: An implicitly restarted lanczos bidiagonalization method with refined harmonic shifts for computing smallest singular triplets. Journal of Computational and Applied Mathematics **260**, 208–217 (2014)

[38] Olney, A.M.: Large-scale latent semantic analysis. Behavior research methods **43**(2), 414–423 (2011)

[39] Paige, C.C., Parlett, B.N., Van der Vorst, H.A.: Approximate solutions and eigenvalue bounds from Krylov subspaces. Numerical linear algebra with applications **2**(2), 115–133 (1995)

[40] Paige, C.C., Saunders, M.A.: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software (TOMS) **8**(1), 43–71 (1982)

[41] Saad, Y.: Variations on Arnoldi's method for computing eigenelements of large unsymmetric matrices. Linear Algebra and its Applications **34**, 269–295 (1980)

[42] Saad, Y.: Numerical Methods for Large Eigenvalue Problems: Revised Edition. Society for Industrial and Applied Mathematics (2011)

[43] Simek, K., Fujarewicz, K., Świerniak, A., Kimmel, M., Jarzab, B., Wiench, M., Rzeszowska, J.: Using SVD and SVM methods for selection, classification, clustering and modeling of DNA microarray data. Engineering Applications of Artificial Intelligence **17**(4), 417–427 (2004)

[44] Simon, H.D., Zha, H.: Low-rank matrix approximation using the Lanczos bidiagonalization process with applications. SIAM Journal on Scientific Computing **21**(6), 2257–2274 (2000)

[45] Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method. SIAM Journal on Matrix Analysis and Applications **13**(1), 357–385 (1992)

[46] Stathopoulos, A.: Locking issues for finding a large number of eigenvectors of hermitian matrices. Tech. rep., Citeseer (2005)

[47] Stewart, G.W.: On the early history of the singular value decomposition. SIAM Review **35**, 551–566 (1993). DOI 10.1137/1035134

[48] Stoll, M.: A Krylov–Schur approach to the truncated SVD. Linear Algebra and its Applications **436**(8), 2795–2806 (2012)

[49] Wall, M., Rechtsteiner, A., Rocha, L.: Singular value decomposition and principal component analysis. In: A practical approach to microarray data analysis, pp. 91–109. Springer (2003)

[50] Wang, Y., Zhu, L.: Research and implementation of SVD in machine learning. In: 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), pp. 471–475. IEEE (2017)

[51] Wu, K., Simon, H.: Dynamic restarting schemes for eigenvalue problems. Tech. rep., Lawrence Berkeley National Lab., CA (US) (1999)

[52] Wu, K., Simon, H.: Thick-restart Lanczos method for large symmetric eigenvalue problems. SIAM Journal on Matrix Analysis and Applications **22**(2), 602–616 (2000)

[53] Wu, L., Romero, E., Stathopoulos, A.: Primme_svds: A high-performance preconditioned svd solver for accurate large-scale computations. SIAM Journal on Scientific Computing **39**(5), S248–S271 (2017)

CHAPTER 4

Conclusions

Results of this thesis have increased the knowledge in the field of numerical linear algebra and led to promising new methods for solving eigenvalue and singular value problems. These new methods have allowed for greater speeds in calculation of the extreme eigenpairs and singular triplets while using very little storage space, and can be applied to pattern recognition, genomics, data visualization, signal classification, and a broad range of other real world problems.

The Hybrid Iterative Refined Method for Computing a Few Extreme Eigenpairs of a Symmetric Matrix from Chapter 2 presented our restarted hybrid method in which we combine thick-restarting with restarting with refined Ritz vectors in a linear combination but using our new iterative process to speed up convergence. We proved that the sequence of iterative refined Ritz vectors will converge and established the ability to implement a linear combination of these iterative refined vectors to improve the process. Our new method was seen to be competitive through numerical examples with other available codes while using limited memory.

Chapter 3, the *Hybrid Iterative Refined Restarted Lanczos Bidiagonalization Methods* takes the hybrid concept in Chapter 2 and extends it from the symmetric eigenvalue problem to the singular value problem. The new hybrid GKL methods use thick-restarting with Ritz vectors or harmonic Ritz vectors combined with a linear combination of iterative refined Ritz vectors, computed either on the normal or augmented system. As these methods do not require factorization of A, they can be applied to not only the seismic sensor work but many other large data problems. When compared with existing MATLAB codes and other publicly available codes, our new GKL hybrid methods were seen to be very competitive, especially when using

very small basis sizes, i.e., limited memory.

Finally, the next steps will be to take the hybrid GKL code and apply them to multiple seismic sensor data sets. There may be a need for method adjustments as well as evaluations of which signals to begin the investigations. Once feature spaces can be expressed using their singular triplets they can be plotted using PCA. This will allow us to test whether the sensor data will cluster like that of the genome data in other studies and apply the desired confidence bounds. We expect the hybrid GKL codes to be fast and compact enough to implement in our sensor processing scheme in real time.

CHAPTER A

Appendix: MATLAB Code

A.1 MATLAB function trreigs(varargin)

```
function varargout = trreigs(varargin)

% TRREIGS: Computes the K extreme eigenvalue and associated eigenvector
% of a N x N symmetric matrix A.
%
% PROGRAM INFORMATION:
% -------------------
%
%  ...  = TRREIGS(A)
%  ...  = TRREIGS('AFUN',N)
%  ...  = TRREIGS(@(x)A(x),N)
%
% The first input value into IRREIGS can be a numeric matrix or a
% function/function handle that returns Y = A*X. If the input matrix
% is a function/function handle the second input value must be
% the size of the matrix A. The the input/output structure of M-file
% Y = AFUN(X,N).
%
% OUTPUT OPTIONS:
% ---------------
%
% I.)   TRREIGS(A)
%       If convergence, displays the K desired eigenvalues.
%
% II.)  D = TRREIGS(A)
%       If convergence, returns K eigenvalues in the vector D.
%
% III.) [V,D] = TRREIGS(A)
%       If convergence, returns a diagonal matrix D that contains the
%       K eigenvalues along the diagonal and the matrix V contains the
%       corresponding orthogonal eigenvectors, such that A*V = V*D.
%       If TRREIGS reaches the maximum number of iterations before convergence
%       then V = [] and D = []. Use output option IV.).
%
% IV.)  [V,D,FLAG] = TRREIGS(A)
```

```
36 %       This option returns the same structural output as (III) plus a two dimensional array FLAG
37 %       that reports if the algorithm converges and the number of matrix-vector
38 %       products. If FLAG(1) = 0 then this implies normal return: all eigenvalues have
39 %       converged. If FLAG(1) = 1 then the maximum number of iterations have been
40 %       reached before all desired eigenvalues have converged. FLAG(2) contains the
41 %       number of matrix-vector products used by the code. If the maximum number of
42 %       iterations are reached (FLAG(1) = 1), then the matrices V and D contain
43 %       the last available approximations for the eigenpairs.
44 %
45 % INPUT OPTIONS:
46 % --------------
47 %
48 %       ... = TRREIGS(A,OPTS) or TRREIGS('AFUN',N,OPTS) or TRREIGS(@(x)A(x),N,OPTS)
49 %       OPTS is a structure containing input parameters. The input parameters can
50 %       be given in any order and can greatly influence convergence rates. The structure OPTS
51 %       may contain some or all of the following input parameters. If parameter OPTS is missing
52 %       or an input parameter in OPTS is not set, default value(s) are used. The string for the
53 %       input parameters
54 %       can contain upper or lower case characters.
55 %
56 %  INPUT PARAMETER      DESCRIPTION
57 %
58 %  OPTS.COEFF        When K > 1, COEFF is used to determine which matrix is
59 %                    use to compute the coefficients for the linear combination
60 %                    of iterative refined vectors. See Section 6 in the
61 %                    reference.
62 %                    coeff = 1 -> matrix (6.5)
63 %                    coeff = 2 -> Add one more term - matrix (6.9) (Default)
64 %                    coeff = 3 -> Add all terms - matrix (6.10)
65 %                    DEFAULT VALUE COEFF = 2
66 %
67 %  OPTS.DELTA1       Used to determine when thick restarting with Ritz vectors can
68 %                    start switching to restarting with iterative refined vectors.
69 %                    The value should be related to the overall convergence tolerance.
70 %                    Therefore, we restrict the input to be in decimal form, 0<= DELTA1 <= 1
71 %                    and check for switching if the maximum residual norm
72 %                    of the desired Ritz pairs is <= TOL^(DELTA1).
73 %                    EXAMPLE: If TOL = 1d-8 and user would like to start switching when
74 %                    convergence
75 %                    has reached 25% of TOL or 1d-2, user inputs .25. The routine computes
```

```
76 %                    TOL^(.25).
77 %                    Too large and only thick-restarted will be done. Too small and stagnation
78 %                    results from using a poor Krylov space.
79 %                    || A*V_RITZ - V_RITZ*D_RITZ ||_2 <= TOL^(DELTA1)*||A||_2.
80 %                    ||A||_2 is approximated by largest absolute Ritz value.
81 %                    DEFAULT VALUE  DELTA1 = 0.1
82 %
83 %  OPTS.DELTA2       Used to determine when the Iterative refined vectors
84 %                    are "close" enough to the Ritz vectors and can be
85 %                    considered for restarting. Too large and only thick-restarted will be done.
86 %                    Too small and stagnation may result from using iterative refined vectors
87 %                    closer to the non-corresponding Ritz vector. See Example 5.2 in reference.
88 %                    (all 1 <= j <= K) |(Refine vectors_j)^T*(Ritz vectors_j)| > DELTA2
89 %                    Must be 0< DELTA2 < 1
90 %                    DEFAULT VALUE  DELTA2 = 0.9
91 %
92 %  OPTS.FLT          Toggle to restrict using restarting with iterative refined Ritz vectors when
93 %                    the current computed iterative refined Ritz values are better than the past
94 %                    iterations best Ritz approximation. This is only used when K=1 and
95 %                    should only be used when M is small. FLT = 1 uses
96 %                    the restriction, FLT= 0 do not use restriction.
97 %                    DEFAULT VALUE FLT = 1 if M > 2 and FLT = 0 when M = 2.
98 %
99 %  OPTS.K            Number of desired eigenvalues.
100 %                    DEFAULT VALUE  K = 1
101 %
102 %  OPTS.M            Number of Lanczos vectors, i.e. size tridiagonal
103 %                    matrix. Currently, full reorthogonalization is
104 %                    used. Large M will increase non-matrix-vector product
105 %                    CPU times.
106 %                    DEFAULT VALUE  M = 2
107 %
108 %  OPTS.MAXIT        Maximum number of iterations, i.e. maximum number of Lanczos restarts.
109 %                    DEFAULT VALUE  MAXIT = 1000
110 %
111 %  OPTS.MAXITREF     Maximum number of iterations used to find iterative
112 %                    refined Ritz values, see Section 5 Algorithm 5.1 in
113 %                    reference. Recommend 10% of MAXIT. Stagnation prevention
114 %                    is in place. Require MAXITREF > =100.
115 %                    DEFAULT VALUE  MAXITREF = 100
```

```
116 %
117 %  OPTS.SIGMA        Two letter string specifying the location of the desired eigenvalues.
118 %                     'LM' or 'LA' Largest Magnitude or Algebraic
119 %                            'SA' Smallest Algebraic
120 %                    DEFAULT VALUE   SIGMA = 'LM'
121 %
122 %  OPTS.TOL          Tolerance used for convergence. Convergence is determined when
123 %                    || A*V - V*D ||_2 <= TOL*||A||_2. ||A||_2 is approximated by
124 %                    largest absolute Ritz value. V and D are either iterative
125 %                    refined or Ritz. If iterative refined V matrix is made orthogonal.
126 %                    See sections 2 and 7 in the reference.
127 %                    DEFAULT VALUE    TOL = SQRT(EPS) (roughly 1d-8)
128 %
129 %  OPTS.V0           A matrix of starting vectors.
130 %                    DEFAULT VALUE  V0 = randn
131 %
132 %  DATE MODIFIED: 12/03/2020
133 %  VER:  1.0
134 %
135 % AUTHORS: J. Baglama, T, Bella, and J. Picucci
136 %
137 % REFERENCE:
138 %  1. Baglama, J, Bella, T, Picucci, J, "An Iterative Method for Computing a Few
139 %     Eigenpairs of a Large Sparse Symmetric Matrix" preprint 2020.
140 %
141 % ************************************************************************
142 % * This MATLAB code is provided to illustrate Algorithm 6.1 and is NOT   *
143 % * optimized for performance or set up for commercial use.               *
144 % * Any use beyond illustrate purposes (e.g. comparison for publications) *
145 % * requires consent of the authors.                                      *
146 % ************************************************************************
147
148 % Too many output arguments requested.
149 if (nargout >= 4), error('ERROR: Too many output arguments.'); end
150
151 %----------------------------%
152 % BEGIN: PARSE INPUT VALUES. %
153 %----------------------------%
154
155 % No input arguments, return help.
```

```
156 if nargin == 0, help trreigs, return, end
157
158 % Get matrix A. Check type (numeric or character) and dimensions.
159 if ischar(varargin{1}) || isa(varargin{1}, 'function_handle')
160    sst_data = 3;
161    if nargin == 1, error('ERROR: Need N (size of matrix A).'); end
162    n = varargin{2};
163    if ~isnumeric(n) || length(n) > 2
164       error('ERROR: Second argument N must be a numeric value.');
165    end
166 else
167    sst_data = 2;
168    n = size(varargin{1},1);
169    if any(size(varargin{1}) ~= n), error('ERROR: Matrix A is not square.'); end
170    if ~isnumeric(varargin{1}), error('ERROR: A must be a numeric matrix.'); end
171 end
172
173 % Square root of machine tolerance used in convergence testing.
174 sqrteps = sqrt(eps);
175
176 % Set all input options to default values.
177 k=1; hybrid = 0; maxit = 1000; m=2;
178 maxitref=100; projt = 0; sigma = 'LM'; tol = sqrteps;
179 delta2 = 0.9; delta1=0.1; coeff = 2; flt = 0; input_flt=[];
180
181 % Preallocate memory for large matrices.
182 v = spalloc(n,m,n*m); f = spalloc(n,1,n);
183
184 % Get input options from the data structure.
185 if nargin > 1 + ischar(varargin{1})
186    options = varargin{sst_data:nargin};
187    names = fieldnames(options);
188    I = strmatch('COEFF',upper(names),'exact');
189    if  ~isempty(I), coeff = getfield(options,names{I}); end
190    I = strmatch('DELTA1',upper(names),'exact');
191    if ~isempty(I), delta1 = getfield(options,names{I}); end
192    I = strmatch('DELTA2',upper(names),'exact');
193    if ~isempty(I), delta2 = getfield(options,names{I}); end
194    I = strmatch('FLT',upper(names),'exact');
195    if  ~isempty(I), input_flt = getfield(options,names{I}); end
```

```
196     I = strmatch('K',upper(names),'exact');
197     if  ~isempty(I), k = getfield(options,names{I}); end
198     I = strmatch('M',upper(names),'exact');
199     if  ~isempty(I), m = getfield(options,names{I}); end
200     I = strmatch('MAXIT',upper(names),'exact');
201     if ~isempty(I), maxit = getfield(options,names{I}); end
202     I = strmatch('MAXITREF',upper(names),'exact');
203     if ~isempty(I), maxitref = getfield(options,names{I}); end
204     I = strmatch('SIGMA',upper(names),'exact');
205     if  ~isempty(I), sigma = upper(getfield(options,names{I})); end
206     I = strmatch('TOL',upper(names),'exact');
207     if ~isempty(I), tol = getfield(options,names{I}); end
208     I = strmatch('V0',upper(names),'exact');
209     if ~isempty(I), v = getfield(options,names{I}); end
210 end
211
212 %************************************************
213 % Check for some input errors in the data structure.
214 % **** This is not an exhaustive check list. ******
215 %************************************************
216
217 % Check that input values are numerical values.
218 if (~isnumeric(coeff)    || ~isnumeric(delta1) || ~isnumeric(delta2)  || ...
219     ~isnumeric(k)        || ~isnumeric(m)      || ~isnumeric(maxit)  || ...
220     ~isnumeric(maxitref) ||  ~isnumeric(tol))
221    error('ERROR: Incorrect type for input value(s) in the structure.');
222 end
223
224 % Check value of COEFF
225 if coeff ~=1 && coeff ~=2 && coeff ~=3 , error('ERROR: COEFF must 1, 2, or 3'); end
226
227 % Check value of DELTA1
228 if delta1<0 || delta1 > 1, error('ERROR: 0<= DELTA1 <= 1'); end
229
230 % Check value of DELTA2
231 if delta2<=0 || delta2 >= 1, error('ERROR: 0< DELTA2 < 1'); end
232
233 % Check value of FLT.
234 if ~isempty(input_flt) % User input.
235   if input_flt ~=0 && input_flt ~=1, error('ERROR: FLT must 0 or 1'); end
```

```
236   flt = input_flt; % set to user input.
237 end
238
239 % Check value of K.
240 if k <= 0,  error('ERROR: K must be a positive value.'); end
241 if k >= m,  error('ERROR: K is too large compared to M.'); end
242
243 % Check value of M.
244 if m  <= 1,  error('ERROR: M must be greater than 1.');  end
245 % Resize Krylov subspace if M (i.e. number of Lanczos vectors) is larger
246 % than N (i.e. the size of the matrix A) then resize to <= N.
247 if m > n, m = n; end
248
249 % Check FLT again.
250 if k > 1, flt = 1; end     % overide input, cannot be used when k > 1.
251 if isempty(input_flt)      % no user input, check if m was changed
252    if m > 2, flt = 1; end  % set to default
253 end
254
255 % Check value of MAXIT
256 if maxit  <= 0,  error('ERROR: MAXIT must be a positive value.'); end
257
258 % Check value of MAXITREF
259 if maxitref  <= 0,   error('ERROR: MAXITREF must be a positive value.'); end
260 if maxitref > maxit, error('ERROR: MAXITREF should be less than MAXIT'); end
261 if maxitref < 100,   error('ERROR: MAXITREF >= 100'); end
262
263 % Check value of SIGMA.
264 if isnumeric(sigma),   error('ERROR: SIGMA must be SA, LM or LA.'); end
265 if length(sigma) ~= 2, error('ERROR: SIGMA must be SA, LM or LA.'); end
266 if ~strcmp(sigma,'LM') && ~strcmp(sigma,'LA') &&  ~strcmp(sigma,'SM') &&
267 ~strcmp(sigma,'SA') error('ERROR: SIGMA must be LM, LA or SA.');
268 end
269 if strcmp(sigma, 'SM')
270     error('SM currently not supported. Recommend user use A\X and
271     LM - output is 1/lambda');
272 end
273
274 % Check value of TOL.
275 % Set tolerance to machine precision if tol < eps.
```

```
276 if tol < eps, tol = eps; warning('Changing TOL to EPS');  end
277
278 % If starting vector V0 is not given then set starting vector V0 to be a
279 % (n x 1) matrix of normally distributed random numbers.
280 if nnz(v) == 0, v = randn(n,1); end
281
282 % Check starting vector V0.
283 if ~isnumeric(v), error('ERROR: Incorrect starting matrix V0.'); end
284 if ((size(v,1) ~= n) || (size(v,2) ~= 1))
285     error('ERROR: Incorrect size of starting matrix V0.');
286 end
287
288 %--------------------------%
289 % END: PARSE INPUT VALUES. %
290 %--------------------------%
291
292 %------------------------------------------------------------%
293 % BEGIN: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
294 %------------------------------------------------------------%
295 % <- DO NOT MODIFY ->
296 dmax=zeros(m,1);      % Initialize array for max. abs. value of all Ritz values
297 if strcmp(sigma,'SA'), dmax = Inf(m,1); end
298 if strcmp(sigma,'LA'), dmax = -Inf(m,1); end
299 f = zeros(n,1);       % Initialize residual vector f.
300 iter = 1;             % Main loop iteration count.
301 r_rf_0=zeros(m,1);    % Used for iterative refined comparsion from last iteration.
302 rztol=tol^(delta1);   % Tolerance for checking on using iterative refined
303 Smax = 0;             % Holds the maximum absolute value of all computed Ritz values.
304 T = zeros(m,m);       % Initialize T matrix.
305 thick=0;              % Indicate if thick restarted is to be used.
306 thk = 2;              % Indicate starting point for Lanczos.
307
308 %------------------------------------------------------------%
309 % END: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
310 %------------------------------------------------------------%
311
312 %-----------------------------------------------%
313 % BEGIN: INITIALIZATION - ONE STEP OF LANCZOS. %
314 %-----------------------------------------------%
315 v(:,1) = v(:,1)/norm(v(:,1));
```

```
v(:,2) = matrixprod(varargin{1},v(:,1),n); mprod=1;
T(1,1)  = v(:,2)'*v(:,1);
v(:,2) = v(:,2) - v(:,1)*T(1,1);
dotv = v(:,2)'*v(:,1); % Reorth step
v(:,2) = v(:,2) - v(:,1)*dotv;
T(1,1) = T(1,1) + dotv;
T(2,1) = norm(v(:,2)); T(1,2) = T(2,1);
if abs(T(2,1)) < eps*abs(T(1,1))
   error('ERROR: V0 caused |T(2,1)| < eps|T(1,1))-> Change V0.');
end
v(:,2) = v(:,2)/T(2,1);
%-------------------------------------------%
% END: INITIALIZATION - ONE STEP OF LANCZOS. %
%-------------------------------------------%

%---------------------------%
% BEGIN: ITERATION PROCESS. %
%---------------------------%
while (iter <= maxit)

  %-------------------------%
  % BEGIN: LANCZOS PROCESS. %
  %-------------------------%
  for j=thk:m
    f = matrixprod(varargin{1},v(:,j),n); mprod=mprod+1;
    if thick == 1
       f = f - v(:,1:j-1)*T(j,1:j-1)';
       thick = 0;
    else
        f = f - v(:,j-1)*T(j,j-1);
    end
    T(j,j) = f'*v(:,j);
    f = f - (v(:,j)*T(j,j));
    dotv = (f'*v(:,1:j))'; % Reorth step
    f = f - (v(:,1:j)*dotv);
    if norm(dotv)>sqrteps        % 2nd reorth step if needed
       dotv2 = (f'*v(:,1:j))';
       dotv=dotv+dotv2;
       f = f - (v(:,1:j)*dotv2);
    end
```

```
356     for i=1:j, T(j,j) = T(j,j) + dotv(i); end
357     fnorm = norm(f);
358     if fnorm < max(Smax*tol,eps) && j>= k && iter > 1
359         T = T(1:j,1:j); m=j; % resize T for exit;
360         warning('Early termination in Lanczos - results may not be trusted');
361         break;
362     end
363     if fnorm < max(Smax*eps,eps) && (j < k || iter == 1)
364         error('ERROR: Lanczos breakdown.');
365     end
366     if j < m
367        v(:,j+1) = f/fnorm;
368        T(j+1,j) = fnorm;
369        T(j,j+1) = T(j+1,j);
370     end
371   end
372   %-----------------------%
373   % END: LANCZOS PROCESS. %
374   %-----------------------%
375
376   %--------------------------------------------------------------------------%
377   % BEGIN: COMPUTE/SORT EIGENVALS AND EIGENVECS OF T AND TEST CONVERGENCE. %
378   %--------------------------------------------------------------------------%
379   [x_rz, d_rz] = eig(T); d_rz = diag(d_rz);
380   Smax = max([Smax;abs(d_rz)]); % Use to app. ||A||_2
381   if strcmp(sigma,'LM')
382      [~,I_eig] = sort(abs(d_rz));
383      I_eig = I_eig(length(I_eig):-1:1);
384      x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
385      for j=1:k
386         dmax(j) = max(dmax(j),abs(d_rz(j)));
387         dmax(j) = sign(d_rz(j))*dmax(j);
388      end
389   elseif strcmp(sigma,'LA')
390      [~,I_eig] = sort(d_rz);
391      I_eig = I_eig(length(I_eig):-1:1);
392      x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
393      for j=1:k
394         dmax(j) = max(dmax(j),d_rz(j));
395      end
```

```
 elseif strcmp(sigma,'SA')
   [~,I_eig] = sort(d_rz);
   x_rz = x_rz(:,I_eig); d_rz = d_rz(I_eig);
    for j=1:k
      dmax(j) = min(dmax(j),d_rz(j));
   end
end

% Convergence check.
nc = 0; % number of convergence desired Ritz values.
res_r = ones(k,1); % reset residual norms.
for j=1:k
    res_r(j) = abs(x_rz(m,j))*fnorm;
    if res_r(j) < tol*Smax, nc = nc+1; end
end
if nc ==k || iter == maxit
   v=v*x_rz(:,1:k); d = diag(d_rz(1:k));
   break;
end

% Compute size matrix T.
Tsz = size(T,1);

%---------------------------------------------------------------------%
% END: COMPUTE/SORT EIGENVALS AND EIGENVECS OF T AND TEST CONVERGENCE. %
%---------------------------------------------------------------------%

%---------------------------------------------------------------------%
% BEGIN: COMPUTE ITERATIVE REFINED AND DETERMINE RESTARTING VECTORS. %
%---------------------------------------------------------------------%

% Compute Iterative refined Ritz values.
rconv=0;
if max(res_r(1)) < rztol*Smax
   [r_rf,x_rf,s_rf,u_rf,rconv] = refined_ritz(dmax(1:k),fnorm,T,maxitref,Tsz,k);
end

% rconv returns 0 or k. 0 indicates not all converged. rconv = k indicates all k
% converged
% Compute inner product between the k converged iterative refined Ritz vector(s)
```

```
% and Ritz vector(s). Also, check if iterative Ritz values are less than
% the previous Ritz values. If so, do not use.
ang_rz_rf = 0; decr = 0;
for i=1:rconv
    ang_rz_rf(i) = abs(x_rz(:,i)'*x_rf(:,i));  % computer the innner prod. to check angle
    if flt % if true - requires iter. refined to be better approx. prev. eig. of T.
       if strcmp(sigma,'LM') && abs(r_rf(i)) < abs(r_rf_0(i)), decr = 1; end
       if strcmp(sigma,'LA') && r_rf(i) < r_rf_0(i), decr = 1;  end
       if strcmp(sigma,'SA') && r_rf(i) > r_rf_0(i), decr = 1; end
    end
    r_rf_0(i) = dmax(i);
end

% Find all inner products between iterative refined and Ritz > delta2
I = find(ang_rz_rf > delta2);

% Convergence check of iterative refined Ritz vector(s)
if length(I) == k
    norm_res=[]; % reset residual norms.
    [q_rf,~] = qr(x_rf,0); % QR to ensure orthogonal vectors
    nr = 0;                % number of iter. ref. converged.
    for j=1:k
      Tq = T*q_rf(:,j) - r_rf(j)*q_rf(:,j);
      norm_res(j) = sqrt(Tq'*Tq + fnorm^2*q_rf(m,j)^2);
      if norm_res(j) < tol*Smax, nr = nr+1; end
    end
    if nr ==k
       v=v*q_rf(:,1:k); d = diag(r_rf(1:k));
       break;
    end
end

% Determine if restart with linear combination of Iterative refined Ritz
% or thick-restarted with Ritz vectors
 if (length(I) == k &&  max(res_r(1)) < rztol*Smax && decr == 0)

    %-----------------------------------------%
    % BEGIN: RESTART ITERATIVE REFINED SECTION %
    %-----------------------------------------%
    % Re-compute residual vector of iterative refined Ritz. Done without QR and is
```

```
476       % different than convergence check since x_rf are not orthogonal.
477       % Needed for restart. res_rf is m x k "small" vector.
478       res_rf=[]; norm_res = []; % reset residual norms.
479       for j=1:k
480           res_rf(:,j) = s_rf(j)*(u_rf(1:m,j) - x_rf(:,j)*(x_rf(:,j)'*u_rf(1:m,j)));
481           norm_res(j) = sqrt(res_rf(:,j)'*res_rf(:,j) + s_rf(j)^2*u_rf(m+1,j)^2);
482       end
483
484       % Initialize the k coefficients c_rf for the linear combination of
485       % the k iterative refined Ritz vectors x_rf.
486       c_rf = ones(length(r_rf),1);
487       if k > 1
488          Bc = zeros(k-1,k); Tem = T(m,1:m); em = zeros(m,1); em(m) = 1;
489
490          %  coeff matrix - (6.5)
491          if coeff == 1
492             Dc = diag(x_rf(m,1:k));
493             for i=1:k-1
494                for j=1:k
495                   Bc(i,j) = r_rf(j)^(i-1);
496                end
497             end
498            Bc = Bc*Dc;
499          end
500
501          % Added one more term coeff matrix - (6.9)
502          if coeff == 2
503             Bc(1,1:k) = x_rf(m,1:k);
504             for i=2:k-1
505                for j=1:k
506                   Bc(i,j) = Tem*x_rf(:,j)*r_rf(j)^(i-2);
507                end
508             end
509          end
510
511          % Use all terms coeff matrix - (6.10)
512          if coeff == 3
513              Bc(1,1:k) = x_rf(m,1:k);  % if k=2 => 1 x 2 matrix end
514              if k >=3                  % if k=3 => 2 x 3 matrix end
515                Bc(2,1:k) = Tem*x_rf(1:m,1:k);
```

```
516              end
517              if k >= 4
518                for i=3:k-1
519                 for j=1:k
520                   Bc(i,j) = Tem*x_rf(:,j)*r_rf(j)^(i-2);
521                   r_sum = 0;
522                   for jj = 3:i
523                       r_sum =r_sum + r_rf(j)^(i-jj)*em'*T^(jj-2)*u_rf(1:m,j);
524                   end
525                     Bc(i,j) = Bc(i,j) + s_rf(j)*r_sum;
526                 end
527                end
528             end
529          end
530          % compute the solution of the k-1 x k homogeneous system using
531          % null. Following code is needed to avoid zero column(s) in Bc
532          % for numerically converged eigenvectors.
533          Iin = []; Iout=[]; % set variables for which columns are used.
534          Bc_max = max(abs(Bc),[],'all');
535             for i=1:k  % search for columns numerically zero.
536               if norm(Bc(:,i),Inf) < sqrt(eps)*Bc_max
537                 Iout = [Iout i]; % references which columns to remove.
538               else
539                 Iin = [Iin i];
540               end
541             end
542             if ~isempty(Iout) % remove numerically zero columns from Bc
543                Bc = Bc(1:k-length(Iout)-1,Iin);
544                c_rf(1:k) = zeros(k,1);
545                c_rf_in = null(Bc); % call Matlab's null to solve system
546                c_rf(Iin) = c_rf_in(:,1);
547                c_rf(Iout) = norm_res(Iout); % Place zero entries back in sol.
548             else
549                c_rf = null(Bc); % call Matlab's null to solve full system
550             end
551          c_rf=c_rf(:,1); % matlab's null can return more than one sol.
552       end
553
554       %-----------------------------------------------------------%
555       % BEGIN: ONE STEP OF LANCZOS WITHOUT MATRIX VECTOR PRODUCT %
```

```
556       % (See section 6 equations (6.2) & (6.12) in reference [1] %
557       %-----------------------------------------------------------%
558       f_rf = 0;
559       % Used for computing the coefficients the residual vectors
560       for j=1:k
561          f_rf = f_rf + c_rf(j)*x_rf(m,j);
562       end
563       % Compute left and right vectors to avoid matrix-vector product
564       % v_left & v_right are  m x 1 "small" vectors.
565       v_left = x_rf*c_rf; v_right = x_rf*(r_rf.*c_rf)+ res_rf*c_rf;
566       v_left_norm = norm(v_left);
567       v_left = v_left/v_left_norm;
568       v_right = v_right/v_left_norm;
569       % Compute the matrix T
570       T = zeros(2,2);
571       % Update Lanczos vectors v(:,1) & v(:,2). f is reused and avoids extra
572       % storage needs.
573       f = v*v_right + (f_rf/v_left_norm)*f;
574       v(:,1) = v*v_left;
575       T(1,1) = f'*v(:,1);
576       f = f - (v(:,1)*T(1,1));
577       dotv = (f'*v(:,1))';  % Reorth step
578       f = f - (v(:,1)*dotv);
579       T(1,1) = T(1,1) + dotv;
580       fnorm = norm(f);
581       v(:,2) = f/fnorm;
582       T(2,1) = fnorm;
583       T(1,2) = T(2,1);
584       thk=2; thick=0; % set values to start Lanczos at step 2
585       %-----------------------------------------------------------%
586       % END: ONE STEP OF LANCZOS WITHOUT MATRIX VECTOR PRODUCT %
587       %-----------------------------------------------------------%
588       %---------------------------------------%
589       % END: RESTART ITERATIVE REFINED SECTION %
590       %---------------------------------------%
591    else
592       %-----------------------------------%
593       % BEGIN: THICK-RESTART RITZ SECTION %
594       %-----------------------------------%
595
```

```
596         % Simple strategy to improve convergence.
597         k_thk = max(floor(nc + (Tsz-nc)/2),k);
598         if Tsz - 1 - k_thk < 0, k_thk = Tsz-1; end
599 
600         % Set up matrices and vectors for thick-restarted
601         v = [v*x_rz(:,1:k_thk) f/fnorm];
602         T = zeros(k_thk+1,k_thk+1);
603         T(1:k_thk+1,1:k_thk) = [diag(d_rz(1:k_thk)); x_rz(m,1:k_thk)*fnorm];
604         T = tril(T,-1) + tril(T)'; thk = k_thk+1; thick=1;
605         %---------------------------------%
606         % END: THICK-RESTART RITZ SECTION %
607         %---------------------------------%
608    end
609    %-------------------------------------------------------------------%
610    % END: COMPUTE ITERATIVE REFINED AND DETERMINE RESTARTING VECTORS. %
611    %-------------------------------------------------------------------%
612 
613  % Update the main iteration loop count.
614  iter = iter+1;
615 
616 end % end main loop
617 %-------------------------%
618 % END: ITERATION PROCESS. %
619 %-------------------------%
620 
621 %----------------------%
622 % BEGIN: OUTPUT RESULTS %
623 %----------------------%
624 
625 % Output option I: Display eigenvalues only.
626 if (nargout == 0)
627     if iter < maxit
628        eigenvalues = diag(d)
629     else
630        eigenvalues =[]
631     end
632 end
633 % Output option II: Set eigenvalues equal to output vector.
634 if (nargout == 1)
635     if iter < maxit
```

```
636         varargout{1} = diag(d);
637     else
638          varargout{1} =[];
639     end
640 end
641
642 % Output option III: Output diagonal matrix of eigenvalues and
643 % corresponding matrix of eigenvectors.
644 if (nargout == 2)
645     if iter < maxit
646        varargout{1} = v;
647        varargout{2} = d;
648     else
649       varargout{1} =[];
650       varargout{2} =[];
651     end
652 end
653
654 % Output option IV: Output diagonal matrix of eigenvalues and
655 % corresponding matrix of eigenvectors and FLAG.
656 FLAG(1) = 0; FLAG(2) = mprod;
657 if iter >= maxit,FLAG(1) = 1; end
658 if nargout == 3
659     varargout{1} = v;
660     varargout{2} = d;
661     varargout{3} = FLAG;
662 end
663
664 %---------------------%
665 % END: OUTPUT RESULTS %
666 %---------------------%
667
668 %--------------------------------------%
669 % BEGIN: COMPUTE REFINED RITZ ITERATION %
670 %--------------------------------------%
671
672 function [rho,v_min,s_min,u_min,rconv] = refined_ritz(D_ritz,R,T,maxitref,Tsz,k)
673 % Computes the Iterative refined Ritz values and vectors. Also, computes any needed vectors
674 % for computing residuals.
675 %
```

```
% Input:
%   D_ritz  - (K x 1) vector of eigenvalues of T (aka Ritz values).
%        R  - real number - norm of residual vector F.
%        T  - (TSZ x TSZ)  Lanczos tridiagonal matrix.
% MAXITREF  - Integer indicating the maximum number of iterations for  the iterative Refined
% Ritz values.
%     TSZ  -  Integer indicates the size of the  tridiagonal matrix.
%
% Output:
%     RHO  - (K x 1) vector of iterative refined Ritz values.
%   V_MIN  - (TSZ x K) Matrix of right singular values of [T; 0 R] and iterative refined Ritz
% vectors.
%   S_MIN  - (K x 1) vector of minumum singular values of [T; 0 R] associated with V_MIN and
% U_MIN.
%            Values are needed in computing residuals.
%   U_MIN  - (TSZ+1 x K) Matrix of left singular values of [T; 0 R]
%            Values are needed in computing residuals.
%   RCONV  - Integer indicate if all K iterative refined Ritz converge.
%
%  Algorithm 5.1 in reference [1].
%
%  DATE MODIFIED: 06/03/2020
%  VER:  1.0

% Set up the augmented matrix [T; 0 R]
T_aug = zeros(Tsz+1,Tsz); T_aug(1:Tsz,1:Tsz) = T;
T_aug(Tsz+1:Tsz+1,Tsz) = R;

% Initialize values.
s_min = zeros(k,1);           % Intialize min. singular values of [T; 0 R].
v_min = zeros(Tsz,k);         % Intialize right singular values of [T; 0 R].
u_min = zeros(Tsz+1,k);       % Intialize left singular values of [T; 0 R].
rconv = zeros(k,1);           % Intialize convergence.
rho = D_ritz;                 % Initialize rho value.
rho_0 = rho;                  % Used to check for convergence during iterations.
sqrteps = sqrt(eps);          % square root of machine precision - eps

% Compute k number of iterative refined Ritz values/vectors
for j = 1:k

```

```
716     %  Set difference in rho values to test for stagnataion.
717     diff_rho_0 = -1; v_min_0 = zeros(Tsz,1);
718
719     % Iteratation to compute the iterative refined Ritz values/vectors
720     for i=1:maxitref
721
722       % Compute the SVD of  [T; 0 R] - rho* I
723       [U,S,V] = svd((T_aug-rho(j)*eye(Tsz+1,Tsz)),0);
724
725       % Need the smallest singular triplet of [T; 0 R] - rho* I. Matlab
726       % returns order of singular values largest to smallest.
727       s_min(j) = S(Tsz,Tsz); v_min(:,j) = V(:,Tsz); u_min(:,j) = U(:,Tsz);
728
729       % Compute the new rho = v_min'*T*v_min (aka refined Ritz value)
730       rho(j) = v_min(:,j)'*T*v_min(:,j);
731
732       % Compute the difference of previous rho to check for convergence
733       diff_rho = abs( (rho_0(j) - rho(j))/rho(j) );
734
735       % Check for convergence
736       if (diff_rho < eps && abs(v_min(:,j)'*u_min(1:Tsz,j)) < sqrteps)...
737          || s_min(j) < eps || norm(v_min_0 - v_min) < eps
738             rconv(j) = 1;  break;
739       end
740
741       % Check for stagnation to avoid too many unecessary iterations. Care
742       % must be taken to avoid the situation where rho(k) is still changing
743       % very slightly at first and then an increase in change later. An
744       % early termination due to stagnation with no converegnce
745       % may avoid increase in change later that converges.
746       if abs(diff_rho_0 - diff_rho)/diff_rho < eps,  break; end  % stagnate
747       if i>= 10 && mod(i,10)==0 % update every 10 iterations to avoid early termination.
748          diff_rho_0 = diff_rho;
749       end
750
751      % Update rho_0 and v_min_0 to test convergence.
752      rho_0(j) = rho(j); v_min_0 = v_min;
753     end
754 end
755
```

```
% After K iteration finish check to see if *all* K iterative Refined Ritz values
% have all converged. Reset rconv to return an integer value, rconv = k all
% converged and rconv = 0, not all converged.
if all(rconv == 1), rconv = k; else, rconv = 0; end

%-------------------------------------%
% END: COMPUTE REFINED RITZ ITERATION %
%-------------------------------------%

%--------------------------------%
% BEGIN: MATRIX-VECTOR PRODUCT. %
%--------------------------------%
function x = matrixprod(A,x,n)
% Computes the matrix vector products.
%
% Input:
%       A  - Matrix A.
%       x  - (N x 1) vector to multiply OP (operator).
%       N  - size of A.
% Output:
%       x  - (N x 1) Product of OP*X (operator).
%
if ischar(A)
    x=feval(A,x,n); % does not accept input parameters
elseif (isa(A, 'function_handle'))
    x=A(x);        % simple function handle
else
    x = A*x;
end
%----------------------------%
% END: MATRIX-VECTOR PRODUCT. %
%----------------------------%


```

A.2 MATLAB function rd2svds(A,m,n,O,k,tol)

```
function [U,S,V,FLAG] = rd2svds(A,m,n,P,k,tol)
%
%  RD2SVDS: Computes the k largest singular value and associated singular vectors
%           of a m x n matrix A such that A*V = U*S and A'*U = V*S, V'*V=I
%           and U'*U = I and S is a diagonal matrix.
%
%  CONVERGENCE: sqrt(|| A*v - u*s||^2 + || A^T*u - v*s ||^2))<= tol*||A||
%               where norm is the 2-norm and ||A|| is approximated by largest
%               singular value of the projected matrix B over all iterations.
%
%  INPUT:
%   A        -  m x n numeric matrix A or an M-file ('Afunc'). If the m x n matrix A
%               is a filename then y = Afunc(x,m,n,'transpose'). If transpose = 'F',
%               then y = A*x. If transpose = 'T', then y = A'*x.
%   m,n      - size of the m x n matrix A
%   P        -  n x 1 "right" starting vector & first column of Lanczos matrix P,
% no vector P = []
%   k        - number of desird singular triplets
%  tol       - user specified tolerance - if k > 1 then tolerance is changed to
%              min(sqrt(eps), 1d-2*tol_user) for 1,..,k-1 and to tol for k
%              singular triplet
%
% OUTPUT:
%  U   - m x k  left approximate singular vectors
%  V   - n x k  right approximate singular vectors
%  S   - k x k  diagonal matrix of singular values
% FLAG - integer output to indicate convergence
%      - 0 all k singular triplets converged within tol*||A||
%      - 1 not all k singular triplets converged - output the < k converged
%        singular triplets and the last iteration approximations.
%
%
%  DATE MODIFIED: 6/22/21
%  VER:  1.0
%
% AUTHORS:
% James Baglama     email: jbaglama@uri.edu
% Vasilije Perovic  email: perovic@uri.edu
% Jennifer Picucci  email: jenniferpicucci@uri.edu
```

```
%
% REFERENCE:
% Baglama, J, Perovic, V, and Picucci, J, "Hybrid Iterative Refined Restarted
% Lanczos Bidiagonalization Method",  2021 submitted Numerical
% Algorithms, preprint: http://www.math.uri.edu/~jbaglama/paper34.pdf

% Initialize values - maxit_outer and maxit_inner can be changed by user.
maxit_outer= 2000; maxit_inner=100; sqrteps = eps^(1/2); f = zeros(n,1); FLAG(1) = 0;
tol_user=tol; V=zeros(n,k); U=zeros(m,k); S=zeros(k); B = zeros(2,2); Q = zeros(m,2);
if k > 1, tol = min(sqrteps,1d-2*tol_user); end; tol = max(tol,eps); numIterRef = 0;

% Begin iter_k for loop
for iter_k = 1:k

   % Set values for each k value.
   iter = 1;  Smax=0; if iter_k == k, tol = max(tol_user,eps); end

   % Set up starting vector P(:,1)
   if isempty(P), P = randn(n,1); end; if iter_k > 1, P = f - V*(f'*V)'; end
   P = [P/norm(P) zeros(n,1)];

   % Matrix-vector product, orthogonalization, and deflation to get Q(:,1)
   if ischar(A), Q(:,1) = feval(A,P(:,1),m,n,'F'); else, Q(:,1) = A*P(:,1); end
   if iter_k > 1, Q(:,1) = Q(:,1) - U*(Q(:,1)'*U)'; end; B(1,1) = norm(Q(:,1));
   Q(:,1) = Q(:,1)*(1/B(1,1));

   % Matrix-vector product, orthogonalization, and deflation to get P(:,2)
   if ischar(A), P(:,2) = feval(A,Q(:,1),m,n,'T'); else, P(:,2) = (Q(:,1)'*A)'; end
   if iter_k > 1, P(:,2) = P(:,2) - V*(P(:,2)'*V)'; end
   P(:,2) = P(:,2) - P(:,1)*B(1,1); B(1,2) = norm(P(:,2)); P(:,2) = P(:,2)*(1/B(1,2));

   % Matrix-vector product, orthogonalization, and deflation to get Q(:,2)
   if ischar(A), Q(:,2) = feval(A,P(:,2),m,n,'F'); else, Q(:,2) = A*P(:,2); end
   if iter_k > 1, Q(:,2) = Q(:,2) - U*(Q(:,2)'*U)'; end
   Q(:,2) =  Q(:,2) - Q(:,1)*B(1,2); B(2,2) = norm(Q(:,2)); Q(:,2) = Q(:,2)*(1/B(2,2));

   % Matrix-vector product, orthogonalization, and deflation to get f
   if ischar(A), f = feval(A,Q(:,2),m,n,'T'); else, f = (Q(:,2)'*A)'; end
   if iter_k > 1, f = f - V*(f'*V)'; end; f = f - P(:,2)*B(2,2);
   f = f - P*(f'*P)'; fnorm = norm(f);
```

```
80
81    % Begin main iter while loop
82    while (iter <= maxit_outer)
83
84        % Compute the largest singular value and associated vector of B.
85        [u_b,s_b,v_b] = svd(B); v_b = v_b(:,1); u_b = u_b(:,1); s_b = s_b(1,1);
86        Smax = max(Smax,s_b);
87
88        % Compute iterative refined Ritz value/vectors.
89        iter_refined = 0; rho = Smax^2; rho_0 = rho; rconv = 0; v_rf_0 = zeros(2,1);
90        T = B'*B; B_aug = [T; 0 B(2,2)*fnorm]; diff_rho_0 = -1;
91        for i=1:maxit_inner
92           B_aug(1,1) = T(1,1) - rho; B_aug(2,2) = T(2,2) - rho;
93           [u_rf,s_rf,v_rf] = svd(B_aug,0);
94           v_rf = v_rf(:,2); s_rf = s_rf(2,2); rho = norm(B*v_rf)^2;
95           diff_rho = abs((rho_0 - rho)/max(rho,rho_0));
96           if (diff_rho < eps && abs(v_rf'*u_rf(1:2,2)) < sqrteps) ...
97           || norm(abs(v_rf_0) - abs(v_rf)) < eps || s_rf < eps
98              u_rf = B*v_rf; alpha = norm(u_rf); u_rf = u_rf/alpha;
99              rho = sqrt(rho); rconv = 1; break; % Converge iter. ref. exit.
100          end
101          if abs(diff_rho_0 - diff_rho)/max(diff_rho,eps) < eps, break; end
102          % Stagnate exit
103          if i >= 10 && mod(i,10) == 0, diff_rho_0 = diff_rho; end
104          rho_0 = rho; v_rf_0 = v_rf;
105       end
106       if abs(v_b'*v_rf) > 0.9 && rconv == 1 && iter>1, iter_refined = 1; end
107
108       % Convergence check.
109       if ~iter_refined
110         B_aug = [-Smax*eye(2,2) B; B' -Smax*eye(2,2); zeros(1,4)]; B_aug(5,2) = fnorm;
111         [~,~,v] = svd(B_aug,0); v_c = v(3:4,4); v_c = v_c/norm(v_c);
112         u_c = v(1:2,4); u_c = u_c/norm(u_c);
113       else
114         v_c = v_rf; u_c = u_rf;
115       end
116       rho_c = u_c'*B*v_c;
117       norm_res = sqrt(norm(B*v_c - rho_c*u_c)^2 + norm(B'*u_c - rho_c*v_c)^2
118       + (u_c(2)*fnorm)^2);
119       if norm_res < tol*max(Smax,S(1,1)) &&  abs(v_b'*v_c) > 0.9, break; end
```

```
120        % Converge exit
121
122        if iter_refined
123
124          % Count number of iter. refined restarts.
125          numIterRef = numIterRef +1;
126
127          % Restart with iterative refined Ritz value/vectors.
128          f = P*(B'*u_rf - alpha*v_rf) + f*u_rf(2); B=[]; B(1,1) = alpha;
129          P(:,1) = P*v_rf; Q(:,1) = Q*u_rf; f = f - (P(:,1)'*f)*P(:,1);
130          B(1,2) = norm(f); P(:,2) = f*(1/B(1,2));
131
132       else
133
134         % Restart with Ritz value/vectors
135         P = [P*v_b f*(1/fnorm)]; Q(:,1) = Q*u_b;
136         B(1,1) = s_b; B(1,2) = fnorm*u_b(2);
137
138       end
139
140       % Matrix-vector product, orthogonalization, and deflation to get Q(:,2)
141       if ischar(A), Q(:,2) = feval(A,P(:,2),m,n,'F'); else, Q(:,2) = A*P(:,2); end
142       if iter_k > 1, Q(:,2) = Q(:,2) - U*(Q(:,2)'*U)'; end
143       Q(:,2) = Q(:,2) - Q(:,1)*(Q(:,2)'*Q(:,1));
144       B(2,2) = norm(Q(:,2)); Q(:,2) = Q(:,2)*(1/B(2,2));
145
146      % Matrix-vector product, orthogonalization, and deflation to get f
147      if ischar(A), f = feval(A,Q(:,2),m,n,'T'); else, f = (Q(:,2)'*A)'; end
148      if iter_k > 1, f = f - V*(f'*V)'; end; f = f - P(:,2)*B(2,2);
149      fnorm = norm(f);
150
151      iter = iter+1;
152
153    end % end: iter while loop
154
155    % Deflation or output.
156    U(:,iter_k)= Q*u_c; V(:,iter_k)= P*v_c; S(iter_k,iter_k) = rho_c;
157    if iter >= maxit_outer
158       U = U(:,1:iter_k); V = V(:,1:iter_k); S = S(1:iter_k,1:iter_k);
159       FLAG(1) = 1; return;
```

```
160     end
161     FLAG(2) = numIterRef;
162
163 end % end: iter_k for loop
164
```

A.3 MATLAB function trrsvds(varargin)

```
function varargout = trrsvds(varargin)

% TRRSVDS: Computes the k largest or smallest singular values and associated
%          singular vectors of a m x n matrix A such that A*V = U*S and
%          A'*U = V*S, V'*V=I and U'*U = I and S is a diagonal matrix.
%
% PROGRAM INFORMATION:
% --------------------
%
%  ...  = TRRSVDS(A)
%  ...  = TRRSVDS('AFUN',m,n)
%
% The first input argument into TRRSVDS can be a numeric matrix A or
% a function. If the m x n matrix A is a function, 'Afunc',
% then the structure must be y = Afunc(x,m,n,'transpose'). If transpose = 'F',
% then y = A*x. If transpose = 'T', then y = A'*x.
%
% OUTPUT OPTIONS:
% ---------------
%
% I.)   TRRSVDS(A)
%       If convergence, displays the k desired singular values.
%
% II.)  S = TRRSVDS(A)
%       If convergence, returns k singular values in the vector S.
%
% III.) [U,S,V] = TRRSVDS(A)
%       If convergence, S is a diagonal matrix that contains the k desired
%       singular values in descending order along the diagonal, the matrix
%       V contains the corresponding "right" singular vectors, and U contains
%       the corresponding "left" singular vectors such that that A*V = U*S,
%       A'*U = V*S, V'*V = I, and U'*U = I. If TRRSVDS reaches the maximum
%       number of iterations before convergence then U=[], S = [] and V = [].
%
% IV.)  [U,S,V,STATS] = TRRSVDS(A)
%       This option returns the same as (III) plus approximation of U,S,V if
%       no convergence and a structure STATS that reports statistical
%       information:
%
```

```
%       STATS =
%       numMatProds:  -> number of matrix-vector products with A and A^T
%       timeMatProds: -> total time computing products with A and A^T
%       numIterRefRestart: -> number of restarts with Iterative Refined vectors
%       timeCompIterRef:   -> time computing iterative Refined vectors
%                             (independent of using them to restart)
%       timeReorth: -> time spent on full reorthogonalization
%       timeTotal:  -> total time elasped
%       estimateSVmax: -> estimate of the maximum singular value over all iterations
%       estimateSVmin: -> estimate of the maximum singular value over all iterations
%       convergedKVals: -> true if all k singular triplets converged, otherwise false
%       outputestRitzorIterRef: -> if converged, names which values were used to exit
%                                  Ritz, Harmonic Ritz, Refined Ritz, or Iterative Refined Ritz.
%
%       If the maximum number of iterations are reached before convergence of all k desired
%       singular values, convergedKVals = 'False' then the matrices U, V, and S contain any
%       singular triplets that have converged plus the last singular triplets
%       approximation for the pairs that have not converged.
%
% INPUT OPTIONS:
% --------------
%
%       ... = TRRSVDS(A,OPTS) or TRRSVDS('AFUN',m,n,OPTS)
%       OPTS is a structure containing input parameters. The input parameters can
%       be given in any order and can greatly influence convergence rates. The structure OPTS
%       may contain some or all of the following input parameters. If parameter OPTS is missing
%       or an input parameter in OPTS is not set, default value(s) are used. The string for the
%       input parameters
%       can contain upper or lower case characters.
%
%  INPUT PARAMETER      DESCRIPTION
%
%  OPTS.RoH          Four letter string ('RITZ' or 'HARM') specifying the use of either Ritz
%                    vectors or harmonic Ritz vectors for thick-restarting.
%                    DEFAULT VALUE    RoH = 'HARM'  if SIGMA = 'SS'
%                                     RoH = 'RITZ'  if SIGMA = 'LS' or if cond(B) > 1/sqrt(eps)
%
%  OPTS.K            Number of desired singular values.
%                    DEFAULT VALUE  K = 1
%
```

```
80 %  OPTS.M_B          Number of Lanczos vectors, i.e. size bidiagonal Lanczos
81 %                    matrix. Full reorthogonalization is used. Large M_B will
82 %                    increase non-matrix-vector product CPU times.
83 %                    DEFAULT VALUE  M_B = 2   if SIGMA = 'LS'
84 %                    DEFAULT VALUE  M_B = 15  if SIGMA = 'SS'
85 %
86 %  OPTS.MAXIT        Maximum number of iterations, i.e. maximum number of restarts.
87 %                    DEFAULT VALUE  MAXIT = 2000
88 %
89 %  OPTS.MAXITREF     Maximum number of iterations used to find iterative
90 %                    refined Ritz singular values.
91 %                    DEFAULT VALUE  MAXITREF = 100
92 %
93 %  OPTS.METHOD       Three letter string ('NOR', 'AUG', or 'THK') to determine which method to
94 %                    use.
95 %                    Hybrid method - 'NOR' or 'AUG'. Determines how iterative refined Ritz
96 %                    values/vectors are computed in the hybrid method.
97 %                      NOR - (Hybrid) Lanczos Bidiagonal decomposition of size m
98 %                            AP_m = Q_mB_m  and A'Q_m = P_m B_m' + f_me_m'
99 %                            Equivalent eigenvalue system
100 %                            A'AP_m = P_m B_m'B_m + B_m(m,m)f_me_m'
101 %                           Compute iterative refined Ritz on [B'*B; 0 B_m(m,m)*norm(f)]
102 %                     AUG - (Hybrid) Equivalent eigenvalue system
103 %                           [0 A; A' 0][Q_m 0; 0 P_m] = [Q_m 0; 0 P_m][0 B; B' 0]
104 %                           +[ 0 0; f_me_m' 0]
105 %                           Compute iterative refined Ritz on [0 B; B' 0; norm(f) 0]
106 %                      THK - (non-Hybrid) no iterative refined Ritz values are computed. Thick-
107 %                            restarted only with either Ritz vectors or harmonic Ritz vectors
108 %                            depending on parameter RoH value.
109 %                    DEFAULT VALUE AUG = 'NOR'
110 %
111 %  OPTS.REORTH       Three letter string ('ONE' or 'TWO') specifying whether to use one-sided
112 %                    ('ONE') full reorthogonalization or two-sided ('TWO'). One-sided is
113 %                    performed only on the "short" vectors. Two-sided orthogonality is always
114 %                    used when cond(A) estimated by cond(B) > 1/sqrt(eps).
115 %                    DEFAULT VALUE  REORTH = 'ONE'
116 %
117 %  OPTS.SIGMA        Two letter string ('LS' or 'SS') specifying the location of the desired
118 %                    singular values.
119 %                    'LS'  Largest singular values and 'SS'  Smallest singular values.
```

```
120 %                    DEFAULT VALUE   SIGMA = 'LS'
121 %
122 %  OPTS.TOL          Tolerance used for convergence. Convergence is determined when
123 %                    MAX (SQRT(|| AV - US||^2 + || A'U - VS ||^2))<= TOL*||A||.
124 %                    Norm is the two norm and [U,S,V] are approximated either by Ritz, harmonic
125 %                    Ritz, Refined Ritz, or Iterative Refined Ritz singular triplets. ||A|| is
126 %                    approximated by largest singular value of all projection matrices.
127 %                    DEFAULT VALUE    TOL = SQRT(EPS) (roughly 1d-8)
128 %
129 %  OPTS.P0           An n x 1 starting vector if m >= n and sigma = 'LS' or 'SS' and an m x 1
130 %                    starting vector if m < n and sigma = 'SS'. P0 is not explicitly created
131 %                    and use first column of matrix V.
132 %                    DEFAULT VALUE  P0 = randn(n,1)
133 %
134 %  DATE MODIFIED: 6/22/21
135 %  VER:  1.0
136 %
137 % AUTHORS:
138 % James Baglama     email: jbaglama@uri.edu
139 % Vasilije Perovic  email: perovic@uri.edu
140 % Jennifer Picucci  email: jenniferpicucci@uri.edu
141 %
142 % REFERENCES:
143 % 1. Baglama, J, Perovic, V, and Picucci, J, "Hybrid Iterative Refined Restarted
144 %    Lanczos Bidiagonalization Method",  2021 submitted Numerical
145 %    Algorithms, preprint: http://www.math.uri.edu/~jbaglama/paper34.pdf
146 % 2. Baglama, J, Bella, T, and Picucci, J, "Hybrid Iterative Refined Method for
147 %    Computing a Few Extreme Eigenpairs of a Symmetric Matrix", SIAM J. Sci.
148 %    Comput. Special session on iterative methods, May 4, 2021.
149 %    https://doi.org/10.1137/20M1344834
150
151 % Start timing count using the MATLAB tic command.
152 tStart = tic;
153
154 % Incorrect number of output arguments requested.
155 if (nargout > 4 || nargout==2 ), error('ERROR: Incorrect number of output arguments.'); end
156
157 %----------------------------%
158 % BEGIN: PARSE INPUT VALUES. %
159 %----------------------------%
```

```
160
161 % No input arguments, return help.
162 if nargin == 0, help trrsvds, return, end
163
164
165 % Matrix A is stored in varargin{1}. Check type (numeric or character) and dimensions.
166 if (isstruct(varargin{1})), error('A must be a matrix.'), end
167 if ischar(varargin{1})
168    if nargin == 1, error('Need dimension M for matrix A).'), end
169    m = varargin{2};
170    if ~isnumeric(m) || length(m) ~= 1
171       error('Second argument M must be a numeric value.');
172    end
173    if nargin == 2, error('Need dimension N for matrix A).'); end
174    n = varargin{3};
175    if ~isnumeric(n) || length(n) ~= 1
176       error('Third argument N must be a numeric value.');
177    end
178 else
179    if ~isnumeric(varargin{1}), error('ERROR: A must be a numeric matrix.'); end
180    [m,n] = size(varargin{1});
181 end
182
183 % Square root of machine tolerance used in convergence testing.
184 sqrteps = sqrt(eps);
185
186 % Set all input options to default values.
187 k=1; maxit = 1000; m_b=[]; reorth='ONE'; method='NOR';
188 maxitref=100; sigma = 'LS'; tol = sqrteps; V=[]; roh = [];
189
190 % Get input options from the data structure.
191 if nargin > 1 + 2*ischar(varargin{1})
192    options = varargin{2+2*ischar(varargin{1}):nargin};
193    names = fieldnames(options);
194    I = strmatch('ROH',upper(names),'exact');
195    if  ~isempty(I), roh = upper(getfield(options,names{I})); end
196    I = strmatch('K',upper(names),'exact');
197    if  ~isempty(I), k = getfield(options,names{I}); end
198    I = strmatch('M_B',upper(names),'exact');
199    if  ~isempty(I), m_b = getfield(options,names{I}); end
```

```
200     I = strmatch('MAXIT',upper(names),'exact');
201     if ~isempty(I), maxit = getfield(options,names{I}); end
202     I = strmatch('MAXITREF',upper(names),'exact');
203     if ~isempty(I), maxitref = getfield(options,names{I}); end
204     I = strmatch('METHOD',upper(names),'exact');
205     if  ~isempty(I), method = upper(getfield(options,names{I})); end
206     I = strmatch('REORTH',upper(names),'exact');
207     if ~isempty(I), reorth = upper(getfield(options,names{I})); end
208     I = strmatch('SIGMA',upper(names),'exact');
209     if  ~isempty(I), sigma = upper(getfield(options,names{I})); end
210     I = strmatch('TOL',upper(names),'exact');
211     if ~isempty(I), tol = getfield(options,names{I}); end
212     I = strmatch('P0',upper(names),'exact');
213     if ~isempty(I), V = getfield(options,names{I}); end
214 end
215
216 %*************************************************
217 % Check for some input errors in the data structure.
218 % **** This is not an exhaustive check list. ******
219 %*************************************************
220
221 % Check that input values are numerical or char values.
222 if (~isnumeric(k) || ~isnumeric(m_b) || ~isnumeric(maxit) || ...
223     ~isnumeric(maxitref) || ~ischar(method)|| ~ischar(reorth) || ...
224     ~ischar(sigma)  || ~isnumeric(tol))
225        error('ERROR: Incorrect type for input value(s) in the structure.');
226 end
227
228 % Check value of MAXIT
229 if maxit  <= 0,  error('ERROR: MAXIT must be a positive value.'); end
230
231 % Check value of MAXITREF
232 if maxitref  <= 0,       error('ERROR: MAXITREF must be a positive value.'); end
233 if maxitref > maxit,     error('ERROR: MAXITREF should be less than MAXIT'); end
234 if maxitref == 1
235      warning(['WARNING: MAXITREF = 1 - computing refined Ritz - not iterative refined
236      Ritz.']);
237      warning(['WARNING: MAXITREF = 1 - not recommended with this routine.']);
238 end
239 if maxitref < 100
```

```
240     warning(['WARNING: MAXITREF should be at least 100 to ensure Iterative Refined Ritz
241     are computed.']);
242 end
243
244 % Check value of METHOD
245 if length(method)  ~= 3, error('ERROR: METHOD must be NOR, AUG, or THK'); end
246 if (~strcmp(method,'NOR') && ~strcmp(method,'AUG') && ~strcmp(method,'THK'))
247    error('ERROR: METHOD must be NOR, AUG, or THK.');
248 end
249
250 % Check the values of REORTH
251 if length(reorth) ~= 3, error('ERROR: REORTH must be ONE or TWO.'); end
252 if (~strcmp(reorth,'ONE') && ~strcmp(reorth,'TWO') )
253    error('ERROR: REORTH must be ONE or TWO.');
254 end
255
256 % Check value of SIGMA.
257 if length(sigma)  ~= 2, error('ERROR: SIGMA must be LS or SS'); end
258 if (~strcmp(sigma,'SS') && ~strcmp(sigma,'LS') )
259    error('ERROR: SIGMA must be LS or SS.');
260 end
261
262 % Interchange m and n so that size(A'A) = min(m,n). Avoids
263 % finding zero values when searching for the smallest singular values.
264 interchange = 0; if n > m && strcmp(sigma,'SS'), t=m; m=n; n=t; interchange = 1; end
265
266 % Determine value of m_b to use
267 if isempty(m_b)
268   if strcmp(sigma,'LS'), m_b = 2; else, m_b = 15; end
269 end
270
271 % Preallocate memory for W and F. These matrices are full and resizing will cause
272 % an increase in cpu time.
273 W = zeros(m,m_b); F = zeros(n,1);
274
275 % If starting p0 is not given then set starting vector p0 to be a
276 % (n x 1) matrix of normally distributed random numbers.
277 % p0 is not explicitly created and use first column of matrix V.
278 if isempty(V)
279   V = zeros(n,m_b); % Preallocate memory for for all V.
```

```
  V(:,1) = randn(n,1);
else
  V(:,2:m_b) = zeros(n,m_b-1); % Preallocate memory for other columns of V.
end

% Check for input errors in the data structure for K, M_B, and TOL
if k       <= 0,   error('ERROR: K must be a positive value.'),  end
if k  >  min(n,m), error('ERROR: K must be less than min(n,m)'), end
if m_b     <= 1,   error('ERROR: M_B must be greater than 1.'),  end
if tol     <  0,   error('ERROR: TOL must be non-negative.'),    end
if m_b >= min(n,m)
   m_b = floor(min(n,m)-0.1);
   warning(['Changing M_B to ',num2str(m_b)]);
   if m_b - k - 1  < 0
      k = m_b - 1;
      warning(['Changing K to ',num2str(k)]);
   end
end
if m_b - k - 1  < 0
   m_b = ceil(k+1+0.1);
   warning(['WARNING: Changing M_B to ',num2str(m_b)]);
end
if m_b >= min(n,m)
   m_b = floor(min(n,m)-0.1);
   k = m_b - 1;
   warning(['WARNING: Changing K to ',num2str(k)]);
   warning(['WARNING: Changing M_B to ',num2str(m_b)]);
end
if ~isnumeric(V), error('ERROR: Incorrect starting vector P0.'), end
if (size(V,1) ~= n), error('ERROR: Incorrect size of starting vector P0.'), end

% Check value of TOL.
% Set tolerance to machine precision if tol < eps.
if tol < eps, tol = eps; warning('WARNING: Changing TOL to EPS');  end

% Determine which vectors to use for  thick-restarting
if isempty(roh)
   if strcmp(sigma,'LS'), roh = 'RITZ'; else, roh = 'HARM'; end
else
  if length(roh) ~= 4, error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM');
```

```
320   end
321   if ~ischar(roh), error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM'); end
322   if (~strcmp(roh,'RITZ') && ~strcmp(roh,'HARM') )
323      error('ERROR: Unknown value for RoH. RoH must be RITZ or HARM');
324   end
325 end
326
327 %--------------------------%
328 % END: PARSE INPUT VALUES. %
329 %--------------------------%
330
331 %-------------------------------------------------------------%
332 % BEGIN: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
333 %-------------------------------------------------------------%
334 % <- DO NOT MODIFY ->
335
336 % Set converegnce values for testing to 0 or empty.
337 conv_rz_tol =0; conv_hm_tol =0; conv_it_tol=0; conv_rz_sqrt=0; conv_hm_sqrt=0;
338 conv_it_sqrt = 0; num_conv_val=0; V_B_cv=[]; U_B_cv=[]; S_B_cv=[]; out_val = -1;
339 res_norm_rz=0;
340
341 % Set all values empty for computing singular triplets for matrix B
342 U_B_rz=[]; S_B_rz=[]; V_B_rz=[];        % Holds the singular triplets of B (aka Ritz)
343 U_B_hm1=[]; S_B_hm1=[];V_B_hm1=[];      % Holds the singular triplets and intermediate
344 R_hm=[]; V_B_hm2=[];V_B_hm_last=[];     % steps for computing harmonic Ritz
345 s=[]; V_B_hm3 =[];V_B_hm=[];S_B_hm=[];  % singular values of B
346 U_rf=[]; S_rf=[]; V_rf=[]; s_min=[];    % Holds iterative refined Ritz singular values of B
347 W_rf=[]; rconv=0; U_rf1=[]; S_rf1=[];   % and values used for convergence tests.
348 V_rf1=[]; s_min1=[]; W_rf1=[]; rconv1=0;
349 U_B_it=[]; S_B_it=[]; V_B_it=[];
350
351 % Initialize array for max/min. singular value of B depending on sigma.
352 if strcmp(sigma,'SS'), dmax = Inf(m_b,1); end
353 if strcmp(sigma,'LS'), dmax = -Inf(m_b,1); end
354
355 % Initialization and description of local variables.
356 B = [];               % Bidiagonal matrix.
357 Bsz =[];              % Size of the bidiagonal matrix (will be <= m_B)
358 delta = 0.9;          % Used to determine when the Iter. refined Ritz singular vectors
359                       % are "close" enough to the desired Ritz singular vectors
```

```
360 eps23 = eps^(2/3);    % Two thirds of eps, used for Smax. Avoids using zero.
361 I=[];                 % Used for indexing.
362 iter = 1;             % Main loop iteration count.
363 k_org = k;            % Set k_org to the original value of k. k may be adjusted during
364                       % iterations.
365 Fnorm =[];            % Two norm of the residual vector F.
366 Ritz = 1;             % Toggle for Lanczos bidiagonalization on which vectors are used
367                       % for restarting
368 Smax = -Inf;          % Holds the maximum value of all computed singular values of B
369                       % est. ||A||_2.
370 Smin = Inf;           % Holds the minimum value of all computed singular values of B
371                       % est. cond(A).
372 SVTol = min(sqrteps,tol);  % Tolerance to determine when Lanczos encounters linearly
373                       % dependent vectors.
374 S_rf_0=zeros(m_b,1);    % Used for iterative refined comparsion from last iteration.
375
376 % Initialization of values used for STATS output.
377 mprod = 0;              % Number of matrix vector products with A and A^T.
378 IterRefRestart = 0;     % Number of restarts with Iter Refined Vectors.
379 timeMatProds=0;         % Total time computing matrix vector products with A and A^T.
380 timeIterRef = 0;        % Total time computing Iter. Ref. values.
381 timeReorth = 0;         % Total time doing full reorthogonalization
382
383 %-------------------------------------------------------------%
384 % END: DESCRIPTION AND INITIALIZATION OF LOCAL VARIABLES. %
385 %-------------------------------------------------------------%
386
387 %---------------------------%
388 % BEGIN: ITERATION PROCESS. %
389 %---------------------------%
390 while (iter <= maxit)
391
392     % Compute the Lanczos bidiagonalization decomposition.
393     [V,W,F,B,mprod,timeMatProds,timeReorth] = ablanzbd(varargin{1},V,...
394      W,F,B,k,interchange,m_b,n,m,mprod,Ritz,SVTol*Smax,reorth,iter,...
395      timeMatProds,timeReorth);
396
397     % Reset k back to the original value k_org.
398     k = k_org;
399
```

```
400     % Determine the size of the bidiagonal matrix B.
401     Bsz = size(B,1);
402
403     % Compute the norm of the vector F.
404     Fnorm = norm(F);
405
406     % Compute singular triplets of B. MATLAB's svds orders the singular values
407     % largest to smallest.
408     [U_B_rz,S_B_rz,V_B_rz] = svd(B); S_B_rz = diag(S_B_rz);
409
410     % Estimate ||A|| using the largest singular value over all iterations
411     % and estimate the cond(A) using approximations to the largest and smallest
412     % singular values. If a small singular value is less than sqrteps use only Ritz
413     % vectors for thick restarted and require two-sided reorthogonalization.
414     if iter==1
415        Smax = S_B_rz(1); Smin = S_B_rz(Bsz);
416     else
417        Smax = max(Smax,S_B_rz(1)); Smin = min(Smin,S_B_rz(Bsz));
418     end
419     Smax = max(eps23,Smax);
420     if Smin/Smax < sqrteps, reorth = 'TWO'; roh='RITZ'; end
421
422     % Re-order the singular values accordingly. MATLAB's SVD orders the
423     % singular values largest to smallest.
424     if strcmp(sigma,'SS')
425         [~,I] = sort(S_B_rz,'ascend');
426         U_B_rz = U_B_rz(:,I); V_B_rz = V_B_rz(:,I); S_B_rz = S_B_rz(I);
427     end
428
429     % If selected compute Harmonic Ritz values and vectors.
430     U_B_hm=[]; S_B_hm=[]; V_B_hm=[]; % Reset to empty.
431     if strcmp(roh,'HARM')
432       R_hm = Fnorm*U_B_rz(Bsz,:);
433       % Update the SVD of B to be the SVD of [B ||F||E_m].
434       [U_B_hm1,S_B_hm1,V_B_hm1] = svd([diag(S_B_rz) R_hm']); S_B_hm1 = diag(S_B_hm1);
435       if strcmp(sigma,'SS') % reorder if looking for smallest.
436          [~,I] = sort(S_B_hm1,'ascend');
437          U_B_hm1 = U_B_hm1(:,I); V_B_hm1 = V_B_hm1(:,I); S_B_hm1 = S_B_hm1(I);
438       end
439       U_B_hm1 = U_B_rz*U_B_hm1; U_B_hm = U_B_hm1;
```

```
440       V_B_hm2 = [[V_B_rz; zeros(1,Bsz)] flipud(eye(Bsz+1,1))]*V_B_hm1;
441       V_B_hm_last = V_B_hm2(Bsz+1,:); % Set equal to the last row of V_B_hm2.
442       s = Fnorm*(B\flipud(eye(Bsz,1)));
443       V_B_hm3 = V_B_hm2(1:Bsz,:) + s*V_B_hm2(Bsz+1,:);
444
445       % Compute the orthogonal harmonic to get the Rayleigh Quotient values
446       % to test for convergence.
447       [V_B_hm,R_Q] = qr(V_B_hm3,0);
448       D = diag(sign(diag(R_Q))); % Compute signs, so diag(R_Q) > 0 and signs of
449       V_B_hm = V_B_hm*D;         % cols. V_B_hm  & V_B_hm3 are same. Note: D*D = I.
450       S_B_hm = diag(U_B_hm'*B*V_B_hm);  % Rayleigh Quotient values
451
452       % Need to reorder the Harmonic values based on Rayleigh Quotient.
453       if strcmp(sigma,'SS')
454          [~,I] = sort(S_B_hm,'ascend');
455       else
456          [~,I] = sort(S_B_hm,'descend');
457       end
458       U_B_hm = U_B_hm(:,I); V_B_hm = V_B_hm(:,I); S_B_hm = S_B_hm(I);
459     end
460
461     % Used later for calling iterative refined function
462     if strcmp(sigma,'LS')
463      for j=1:k
464         dmax(j) = max([dmax(j),S_B_rz(j)]);
465      end
466     else
467       for j=1:k
468         dmax(j) = min([dmax(j),S_B_rz(j)]);
469       end
470     end
471
472     % Convergence tests for Ritz and if computed Harmonic Ritz singular values/vectors.
473     res_norm_rz_pre = res_norm_rz;
474     [conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt,V_B_cv,
475     U_B_cv,S_B_cv,...
476     res_norm_rz,res_norm_hm,res_norm_it,out_val] = convtests(B,Bsz,Fnorm,tol,k_org,
477     U_B_rz,...
478     S_B_rz,V_B_rz,U_B_hm,S_B_hm,V_B_hm,[],[],[],Smax,sqrteps);
479     num_conv_val = max([num_conv_val,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt]);
```

```
480
481     % If all desired singular values converged then exit main loop.
482     if conv_rz_tol >= k_org || conv_hm_tol >= k_org || iter >= maxit, break,  end
483
484     %--------------------------------------------------------------------%
485     % BEGIN: COMPUTE ITERATIVE REFINED AND DETERMINE RESTARTING VECTORS. %
486     %--------------------------------------------------------------------%
487
488     % Compute Iterative refined Ritz values.
489     rconv = 0; % set the number of convergence iterative refined Ritz to 0.
490     if ~strcmp(method,'THK') && iter > 1
491        IRstart = tic;
492        [U_rf,S_rf,V_rf,rconv] = refined_ritz(dmax(1:k),Fnorm,abs(B(Bsz,Bsz)),B,maxitref,Bsz,
493        k,method);
494        timeIterRef = timeIterRef + toc(IRstart);
495     end
496
497     % rconv returns 0 or k. 0 indicates not all converged. rconv = k indicates all k converged
498     % Compute inner product between the k converged iterative refined Ritz vector(s)
499     % and Ritz vector(s). Also, check if iterative refined Ritz values are less than
500     % the previous Ritz values. If so, do not use iterative refined Ritz
501     % vectors to restart.
502     ang_rz_rf = 0; decr = 0;
503     if rconv > 0
504         ang_rz_rf = abs(diag(V_B_rz(:,1:k)'*V_rf(:,1:k))); % compute the innner prod. to check
505         % angle
506         for i=1:k_org
507            if k_org > 1 || (k_org == 1 && Bsz > 2) % Requires iter. refined to be better approx.
508            prev. vals.
509              if strcmp(sigma,'LS') && (S_rf(i) + eps < S_rf_0(i)), decr = 1;  end
510              if strcmp(sigma,'SS') && (S_rf(i) > S_rf_0(i) + eps),  decr = 1; end
511            end
512         end
513     end
514
515     % Reset S_rf_0 to be prev. dmax values for next iteration comparision
516     S_rf_0 = dmax;
517
518     % Find all inner products between iterative refined Ritz and Ritz >
519     % delta = 0.9
```

```
520     I = find(ang_rz_rf > delta);
521
522     % Convergence check.
523     if length(I) == k && decr == 0
524        U_rf1 = U_rf; V_rf1 = V_rf; % if converged use iterative refined Ritz
525        iterrefined = 1;
526     else
527        % Use refined Ritz on augmented system - when all iterative refined Ritz
528        % failed to converge.
529        [U_rf1,S_rf1,V_rf1,rconv1] = refined_ritz(dmax(1:k),Fnorm,abs(B(Bsz,Bsz)),B,1,Bsz,k,'AUG');
530        iterrefined = 0;
531     end
532
533     [V_B_it,V_R] = qr(V_rf1(:,1:k),0); % QR to ensure orthogonal vectors
534     D = diag(sign(diag(V_R)));         % Compute signs and set diag(R) > 0
535     V_B_it = V_B_it*D;                 % cols.V_B_it & V_rf1 are same. Note: D*D = I
536     [U_B_it,U_R] = qr(U_rf1(:,1:k),0); % QR to ensure orthogonal vectors
537     D = diag(sign(diag(U_R)));         % Compute signs and set diag(R) > 0
538     U_B_it = U_B_it*D;                 % cols. U_B_it & U_rf1 are same. Note: D*D = I
539     S_B_it = diag(U_B_it'*B*V_B_it);   % Rayleigh Quotient values - no need to reorder
540
541     % Double check vectors are close to Ritz vectors.
542     [~,R_ch] = qr(V_B_rz(:,1:k)'*V_B_it(:,1:k),0);
543     if abs(prod(diag(R_ch))) > sqrteps
544       % Convergence tests for Ritz singular value and to determine if to compute Iter. Refined.
545       [conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt,V_B_cv,U_B_cv,
546       S_B_cv,...
547        res_norm_rz,res_norm_hm,res_norm_it,out_val] = convtests(B,Bsz,Fnorm,tol,k,U_B_rz,...
548        S_B_rz,V_B_rz,[],[],[],U_B_it,S_B_it,V_B_it,Smax,sqrteps);
549       num_conv_val = max([num_conv_val,conv_rz_sqrt,conv_hm_sqrt,conv_it_sqrt]);
550     end
551
552     % If all desired singular values converged then exit main loop.
553     if conv_it_tol >= k_org, break, end
554
555     % Do not restart with iterative refined Ritz vectors if no change
556     % bewtween input and output values.
557     if rconv > 0
558       all_conv = 0;
559       for i=1:k_org
```

```
560           if abs(S_rf(i) - dmax(i)) < eps*10, all_conv = all_conv +1; end
561         end
562         if all_conv >= k_org, rconv = 0; end
563       end
564
565       % Do not restart consecutively with iterative refined Ritz vectors if last restart with
566       % iterative refined Ritz vectors caused norm of Ritz values/vectors to increase.
567       if length(I) == k &&  rconv > 0 && decr == 0 && ~strcmp(method,'THK') && Ritz == 0
568           if max(res_norm_rz) < max(res_norm_rz_pre), Ritz = 1; end
569       end
570
571       % Check to determine if restart with iterative refined Ritz or thick-restart
572       if length(I) == k && rconv > 0 && decr == 0 && ~strcmp(method,'THK') && (Ritz ~= 0
573       || k == 1)
574
575         % Count the number of restarts with Iterative Refined.
576         IterRefRestart = IterRefRestart + 1;
577
578         %-----------------------------------------%
579         % BEGIN: RESTART ITERATIVE REFINED SECTION %
580         %-----------------------------------------%
581
582         % Find the coefficients c_rf to setp linear combination.
583         % See reference [1] for details.
584         %
585         % Follows the equivalent eigenvalue problem.
586         % A^T*AV = V*(B^T*B) + B(Bsz,Bsz)*f*e^T
587         % Therefore, the approximate eigenpair is (S_rf^2,V_rf)
588
589         % Re-compute residual error of iterative refined vectors.
590         c_rf = ones(length(S_rf),1);
591         if k > 1
592
593            % For restarting need the square of singular value - i.e. eigenvalue of T = B'*B.
594            S_rf = S_rf.^2; T = B'*B;
595
596            % Compute norm of residual for A^T*AV = V*(B^T*B) + B(Bsz,Bsz)*f*e^T
597            for i=1:k
598               res_norm_it(i,1) = sqrt(norm((T - S_rf(i)*eye(Bsz))*V_rf(:,i))^2 + (abs(B(Bsz,Bsz))
599               *Fnorm*abs(V_rf(Bsz,i)))^2);
```

```
600          end
601
602          % Initialize the k coefficients c_rf for the linear combination of
603          % the k iterative refined Ritz singlar vectors V_rf.
604          Bc = zeros(k-1,k); Tem = T(Bsz,1:Bsz); Bc(1,1:k) = V_rf(Bsz,1:k);
605          for i=2:k-1
606             for j=1:k
607                 Bc(i,j) = Tem*V_rf(:,j)*S_rf(j)^(i-2);
608              end
609          end
610
611          % Compute the solution of the k-1 x k homogeneous system using
612          % null. Following code is needed to avoid zero column(s) in Bc
613          % for numerically converged eigenvectors of B'*B.
614          Iin = []; Iout=[]; % set variables for which columns are used.
615          Bc_max = max(abs(Bc),[],'all');
616          for i=1:k  % search for columns numerically zero.
617             if norm(Bc(:,i),Inf) < sqrteps*Bc_max
618               Iout = [Iout i]; % references which columns to remove.
619             else
620               Iin = [Iin i];
621             end
622          end
623          if ~isempty(Iout) % remove numerically zero columns from Bc
624             Bc = Bc(1:k-length(Iout)-1,Iin);
625             if ~isempty(Bc)
626                c_rf(1:k) = zeros(k,1);
627                c_rf_in = null(Bc); % call Matlab's null to solve system
628                if ~isempty(c_rf_in), c_rf(Iin) = c_rf_in(:,1); end
629                c_rf(Iout) = res_norm_it(Iout); % Place zero entries back in sol.
630             else
631                c_rf = res_norm_it(1:k);
632             end
633          else
634             c_rf = null(Bc); % call Matlab's null to solve full system
635          end
636          c_rf = c_rf(:,1); % matlab's null can return more than one sol.
637       end
638       % The coefficents c_rf have been computed at this point.
639
```

```
640       % Compute the starting vector
641       % v_bar = c_rf(1)*V_rf(:,1)+ .... + c_rf(k)*V_rf(:,k)
642       v_bar = V_rf*c_rf; v_bar = v_bar/norm(v_bar);
643
644       % Compute u_bar = B*v_bar and B(1,1).
645       % Now have A*V*v_bar = W*u_bar*alpha (avoids computing A*V*v_bar)
646       u_bar = B*v_bar; alpha = norm(u_bar); u_bar = u_bar/alpha;
647
648       % Compute F = A^T*W*u_bar - alpha without using A^T*W*u_bar
649       F = V*(B'*u_bar - alpha*v_bar) + F*u_bar(Bsz);
650
651       % Reset B matrix.
652       B=[]; B(1,1) = alpha;
653
654       % Updated first column of V. Do not resize - slows down code.
655       V(:,1) = V*v_bar;
656
657       % Updated first column of W. Do not resize - slows down code.
658       W(:,1) = W*u_bar;
659
660       % Reorthogonalize the residual vector (indepdendent of REORTH)
661       ORstart = tic;
662       F = F - (V(:,1)'*F)*V(:,1);
663       timeReorth = timeReorth + toc(ORstart);
664
665       % Compute the normn of F, scale and update V and B matrices for restarting.
666       beta = norm(F); V(:,2) = F/beta; B(1,2) = beta;
667
668       % Set the Ritz for which vectors restarting
669       % Lanczos bidiagonalization decomposition
670       % Ritz = 0 => iterative refined Ritz vectors
671       % Ritz = 1 => thick-restarting with harmonic or Ritz
672       Ritz = 0;
673       %-----------------------------------------%
674       % END: RESTART ITERATIVE REFINED SECTION %
675       %-----------------------------------------%
676     else
677       %-----------------------------------%
678       % BEGIN: THICK-RESTART RITZ SECTION %
679       %-----------------------------------%
```

```
680
681         % Simple strategy to improve convergence. Adjust k value.
682         k = k_org + num_conv_val;
683         if k < Bsz-1 && k > 1 % used for small values of Bsz
684             if abs(S_B_rz(k+1) - S_B_rz(k)) < abs(S_B_rz(k-1) - S_B_rz(k))
685                 k = k+1;
686             end
687         end
688         k = max(floor((Bsz+num_conv_val)/2),k); % used for large Bsz
689         if k >= Bsz, k = Bsz - 1; end
690
691         % Use Harmonic Ritz vectors for thick-restarting
692         if strcmp(roh,'HARM')
693             % Vectors are not orthogonal.
694             [V_B_hm,R] = qr([ [V_B_hm3(:,1:k); zeros(1,k)]  [-s; 1] ],0);
695             V(:,1:k+1) = [V F/Fnorm]*V_B_hm;
696
697             % Update and compute the K x K+1 part of B.
698             B = diag(S_B_hm1(1:k))*triu((R(1:k+1,1:k)+ R(:,k+1)*V_B_hm_last(1:k))');
699
700            % Update W
701            W(:,1:k) = W*U_B_hm1(:,1:k);
702
703         else
704
705            % Use Ritz vectors for thick-restarting
706            R = Fnorm*U_B_rz(Bsz,:); F = F/Fnorm;
707            V(:,1:k+1) = [V*V_B_rz(:,1:k) F];
708            B = [diag(S_B_rz(1:k)), R(1:k)'];
709
710            % Update W
711            W(:,1:k) = W*U_B_rz(:,1:k);
712
713         end
714
715         % Set the Ritz for which vectors restarting
716         % Lanczos bidiagonalization decomposition
717         % Ritz = 0 => iterative refined Ritz vectors
718         % Ritz = 1 => thick-restarting with harmonic or Ritz
719         Ritz = 1;
```

```
720
721         %---------------------------------%
722         % END: THICK-RESTART RITZ SECTION %
723         %---------------------------------%
724       end
725
726  % Update the main iteration loop count.
727  iter = iter+1;
728
729 end % end main loop
730 %-------------------------%
731 % END: ITERATION PROCESS. %
732 %-------------------------%
733
734 %-----------------------%
735 % BEGIN: OUTPUT RESULTS %
736 %-----------------------%
737
738 % Output option I: Display singular values only.
739 if (nargout == 0)
740     if iter >= maxit
741        disp('Maximum number of iterations exceeded.');
742        disp('Use option IV for best estimate.');
743        SingularValues=[]
744     else
745       SingularValues = S_B_cv(1:k_org)
746     end
747 end
748
749 % Output option II: Set singular values equal to output vector.
750 if (nargout == 1)
751     if iter >= maxit
752       disp('Maximum number of iterations exceeded.');
753       disp('Use option IV for best estimate.');
754       varargout{1}=[];
755     else
756       varargout{1} = S_B_cv;
757     end
758 end
759
```

```
% Output option III and IV: Output singular triplets (U,S,V)
if nargout > 1
    if iter >= maxit && nargout == 3
       disp('Maximum number of iterations exceeded.');
       disp('Use option IV for best estimate.');
       varargout{1}=[]; varargout{2}=[]; varargout{3}=[];
    else
       if interchange
         varargout{1} = V*V_B_cv(:,1:k_org);
         varargout{2} = diag(S_B_cv(1:k_org));
         varargout{3} = W*U_B_cv(:,1:k_org);
       else
         varargout{1} = W*U_B_cv(:,1:k_org);
         varargout{2} = diag(S_B_cv(1:k_org));
         varargout{3} = V*V_B_cv(:,1:k_org);
       end
    end
    % Output options IV: Output singular triplets (U,S,V) and STATS
   if nargout == 4
      STATS.numMatProds = mprod;
      STATS.timeMatProds = timeMatProds;
      STATS.numIterRefRestart = IterRefRestart;
      STATS.timeCompIterRef = timeIterRef;
      STATS.timeReorth = timeReorth;
      STATS.timeTotal = toc(tStart);
      STATS.estimateSVmax = Smax;
      STATS.estimateSVmin = Smin;
      if iter >= maxit
         STATS.convergedKVals='FALSE';
      else
         STATS.convergedKVals='TRUE';
      end
      if out_val == 1
        STATS.outputestRitzorIterRef='RITZ SING.';
        STATS.maxnormres = max(res_norm_rz);
      elseif out_val == 2
          STATS.outputestRitzorIterRef='HARM SING.';
        STATS.maxnormres = max(res_norm_hm);
      elseif out_val == 3
        if iterrefined
```

```
800             STATS.outputestRitzorIterRef='ITER. REF. SING';
801           else
802             STATS.outputestRitzorIterRef='REF. SING';
803           end
804           STATS.maxnormres = max(res_norm_it);
805         elseif out_val == -1
806             STATS.outputestRitzorIterRef='ERROR';
807             STATS.maxnormres = 'ERROR';
808         end
809         varargout{4}=STATS;
810     end
811 end
812
813 %---------------------%
814 % END: OUTPUT RESULTS %
815 %---------------------%
816 end
817
818 %-----------------------------------------------%
819 % BEGIN: LANCZOS BIDIAGONALIZATION DECOMPOSITION %
820 %-----------------------------------------------%
821
822 function [V,W,F,B,mprod,timeMatProds,timeReorth] = ablanzbd(A,V,W,F,B,K,interchange,
823 m_b,n,m,mprod,Ritz,SVTol,reorth,iter,timeMatProds,timeReorth)
824 % Computes the Lanczos bidiagonalization decomposition
825 %
826 %  A*V  = W*B
827 %  A'*W = V*B' + F*E^T
828 %
829 % with full reorthogonalization.
830 % If the m x n matrix A is a function, 'Afunc',
831 % then the structure must be y = Afunc(x,m,n,'transpose'). If transpose = 'F',
832 % then y = A*x. If transpose = 'T', then y = A'*x.
833
834 % James Baglama
835 % DATE: 6/22/21
836
837 % Initialization of main loop count J.
838 J = 1;
839
```

```
840 % Normalize starting vector.
841 if iter == 1
842    V(:,1) = V(:,1)/norm(V(:,1)); B=[];
843 else
844    if Ritz, J = K+1; else, J = 2; end
845 end
846
847 % Matrix A product with vector(s) V, (W=A*V).
848 Astart = tic;
849 if interchange
850    if ischar(A)
851       W(:,J) = feval(A,V(:,J),m,n,'T');
852   else
853       W(:,J) = (V(:,J)'*A)';
854   end
855 else
856   if ischar(A)
857       W(:,J) = feval(A,V(:,J),m,n,'F');
858   else
859       W(:,J) = A*V(:,J);
860   end
861 end
862 timeMatProds = timeMatProds + toc(Astart);
863
864 % Count the number of matrix vector products.
865 mprod = mprod + 1;
866
867 % Input vectors are singular vectors and AV(:,J) which must be orthogonalized.
868 if iter ~= 1
869     W(:,J) = orthog(W(:,J),W(:,1:J-1));
870
871    % Second orthogonalization step required. Input vectors not always
872    % strongly orthogonal.
873    W(:,J) = orthog(W(:,J),W(:,1:J-1));
874
875 end
876
877 % Compute the norm of W.
878 S = norm(W(:,J));
879
```

```
880 % Check for linearly dependent vectors.
881 if S <= SVTol
882    W(:,J) = randn(size(W,1),1);
883    W(:,J) = orthog(W(:,J),W(:,1:J-1));
884    W(:,J) = W(:,J)/norm(W(:,J));
885    S = 0;
886 else
887    W(:,J) = W(:,J)/S;
888 end
889
890 % Begin of main iteration loop for the block Lanczos bidiagonalization decomposition.
891 while (J <= m_b)
892
893    % Matrix A' product with vector(s), (F = A'*W).
894    Astart = tic;
895    if interchange
896       if ischar(A)
897          F = feval(A,W(:,J),m,n,'F');
898       else
899          F = A*W(:,J);
900       end
901    else
902       if ischar(A)
903         F = feval(A,W(:,J),m,n,'T');
904       else
905         F = (W(:,J)'*A)';
906       end
907    end
908    timeMatProds = timeMatProds + toc(Astart);
909
910    % Count the number of matrix vector products.
911    mprod = mprod + 1;
912
913    % One step of the Gram-Schmidt process.
914    F = F - V(:,J)*S;
915
916    % Second step to maintain strong local orthogonality
917    S2 = F'*V(:,J); F = F - V(:,J)*S2; S = S + S2;
918
919    % Always perform full reorthogonalization step of "Short vectors".
```

```
920     ORstart = tic;
921     if J > 1, F = orthog(F,V(:,1:J-1)); end
922     timeReorth = timeReorth + toc(ORstart);
923
924     if J+1 <= m_b
925
926        % Compute the norm of F.
927        R = norm(F);
928
929        % Check for linearly dependent vectors.
930        if R <= SVTol
931           F = randn(size(V,1),1);
932           F = orthog(F,V(:,1:J));
933           V(:,J+1) = F/norm(F);
934           R = 0;
935        else
936          V(:,J+1) = F/R;
937        end
938
939        % Compute bidiagonal matrix B.
940        if isempty(B)
941           B = [S R];
942        else
943           B = [B zeros(J-1,1); zeros(1,J-1) S R];
944        end
945
946        % Matrix A product with vector(s), (W=A*V).
947        Astart = tic;
948        if interchange
949           if ischar(A)
950             W(:,J+1) = feval(A,V(:,J+1),m,n,'T');
951           else
952             W(:,J+1) = (V(:,J+1)'*A)';
953           end
954        else
955           if ischar(A)
956              W(:,J+1) = feval(A,V(:,J+1),m,n,'F');
957           else
958              W(:,J+1) = A*V(:,J+1);
959           end
```

```
960        end
961        timeMatProds = timeMatProds + toc(Astart);
962
963        % Count the number of matrix vector products.
964        mprod = mprod + 1;
965
966        % One step of the Gram-Schmidt process.
967        W(:,J+1) =  W(:,J+1) - W(:,J)*R;
968
969        % Second step for local orthogonality
970        R2 = W(:,J+1)'*W(:,J); W(:,J+1) = W(:,J+1) - W(:,J)*R2; R = R + R2;
971
972        % Full Reorthogonalization step. "Long vectors"
973        ORstart = tic;
974        if ( iter == 1 || strcmp(reorth,'TWO') )
975           if J > 1
976             W(:,J+1) = orthog(W(:,J+1),W(:,1:J-1));
977           end
978        end
979        timeReorth = timeReorth + toc(ORstart);
980
981        % Compute the norm of W.
982        S = norm(W(:,J+1));
983
984       % Check for linearly dependent vectors.
985       if S <= SVTol
986          W(:,J+1) = randn(size(W,1),1);
987          W(:,J+1) = orthog(W(:,J+1),W(:,1:J));
988          W(:,J+1) = W(:,J+1)/norm(W(:,J+1));
989          S = 0;
990       else
991          W(:,J+1) = W(:,J+1)/S;
992       end
993
994     else
995
996      % Add last element to matrix B
997       B = [B; zeros(1,J-1) S];
998
999     end
```

```
1000
1001      % Update iteration count.
1002      J = J + 1;
1003
1004  end
1005  end
1006
1007  function y = orthog(y,X)
1008  % Simple re-orthogonalization of vector y against columns of matrix X.
1009  for i=1:size(X,2)
1010    dotY = y'*X(:,i);
1011    y = y - X(:,i)*dotY;
1012  end
1013
1014  end
1015
1016  %-----------------------------------------------%
1017  % END: LANCZOS BIDIAGONALIZATION DECOMPOSITION %
1018  %-----------------------------------------------%
1019
1020  %--------------------------%
1021  % BEGIN: CONVERGENCE TESTS %
1022  %--------------------------%
1023
1024  function [conv_rz_tol,conv_hm_tol,conv_it_tol,conv_rz_sqrt,conv_hm_sqrt,...
1025            conv_it_sqrt,V_B_cv,U_B_cv,S_B_cv,res_norm_rz,res_norm_hm,res_norm_it,
1026            out_val] = ...
1027            convtests(B,Bsz,Fnorm,tol,k,U_B_rz,S_B_rz,V_B_rz,U_B_hm,...
1028            S_B_hm,V_B_hm,U_B_it,S_B_it,V_B_it,Smax,sqrteps)
1029  % This function checks the convergence of singular triplets
1030
1031  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1032  % Convergence check for all input values.                                   %
1033  %                                                                          %
1034  % SQRT ( || A v - s u ||^2 + || A^T u - s v ||^2 )<= TOL*||A||_2           %
1035  %                                                                          %
1036  % Use the Lanczos bidiagonalization decomposition relationship             %
1037  %  A V   = W B                                                             %
1038  %  A^T W = V B^T + F E^T                                                   %
1039  %                                                                          %
```

```
% || A v - s u ||^2 = ||A V v - s W u||^2 = || W B v - s W u ||^2  %
%                                      = || B v - s u ||^2      %
%                                                               %
% || A^T u - s v ||^2 = ||A^T W u - s V v||^2                    %
%                     = ||V B^T u + F E^T u - s V v||^2          %
% (since V^TF = 0)    = ||B^T u -s v||^2 + ||F E^T u||^2         %
%                     = ||B^T u -s v||^2 + (| E^T u| ||F||)^2    %
%                                                               %
% RITZ => B v = s u and B^T u = s v and                         %
% || A v - s u ||   = 0                                         %
% || A^T u - s v || = | E^T u| ||F||                            %
% Therefore,                                                    %
%                                                               %
% SQRT( || A v - s u ||^2 + || A^T u - s v ||^2 ) = | E^T u| ||F|| %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% James Baglama
% DATE: 3/30/21

% Initialize output values.
conv_rz_tol = 0;  conv_hm_tol=0;  conv_it_tol=0;
conv_rz_sqrt = 0; conv_hm_sqrt=0; conv_it_sqrt=0;
V_B_cv=[]; U_B_cv=[]; S_B_cv=[];
res_norm_hm = ones(k,1); res_norm_it = ones(k,1);

% Compute the residual for Ritz values
res_norm_rz = (abs(U_B_rz(Bsz,1:k))*Fnorm)';
for i=1:k
   if res_norm_rz(i,1) < tol*Smax,  conv_rz_tol  = conv_rz_tol+1;  end
   if res_norm_rz(i,1) < sqrteps*Smax, conv_rz_sqrt = conv_rz_sqrt+1; end
end

% Compute the residual for Harmonic Ritz values.
if ~isempty(S_B_hm)
  for i=1:k
    res_norm_1(i,1) = norm(B*V_B_hm(:,i) - S_B_hm(i)*U_B_hm(:,i))^2;
    res_norm_2(i,1) = norm(B'*U_B_hm(:,i) - S_B_hm(i)*V_B_hm(:,i))^2 + (U_B_hm(Bsz,i)
    *Fnorm)^2;
    res_norm_hm(i,1) = sqrt(res_norm_1(i,1)+res_norm_2(i,1));
```

```
1080      if res_norm_hm(i,1) < tol*Smax,  conv_hm_tol  = conv_hm_tol+1;  end
1081      if res_norm_hm(i,1) < sqrteps*Smax, conv_hm_sqrt = conv_hm_sqrt+1; end
1082    end
1083  end
1084
1085  % Compute the residual for Itertive Refined Ritz values.
1086  if ~isempty(S_B_it)
1087    for i=1:k
1088      res_norm_1(i,1) = norm(B*V_B_it(:,i) - S_B_it(i)*U_B_it(:,i))^2;
1089      res_norm_2(i,1) = norm(B'*U_B_it(:,i) - S_B_it(i)*V_B_it(:,i))^2 + (U_B_it(Bsz,i)
1090      *Fnorm)^2;
1091      res_norm_it(i,1) = sqrt(res_norm_1(i,1)+res_norm_2(i,1));
1092      if res_norm_it(i,1) < tol*Smax,  conv_it_tol  = conv_it_tol+1;  end
1093      if res_norm_it(i,1) < sqrteps*Smax, conv_it_sqrt = conv_it_sqrt+1; end
1094    end
1095  end
1096
1097  % Output values.
1098  [~,I] = max([conv_rz_tol conv_hm_tol conv_it_tol]);
1099  if I == 1, out_val = 1; U_B_cv = U_B_rz; V_B_cv= V_B_rz; S_B_cv = S_B_rz; end
1100  % Output Ritz (Default)
1101  if I == 2, out_val = 2; U_B_cv = U_B_hm; V_B_cv= V_B_hm; S_B_cv = S_B_hm; end
1102  % Output Harmonic
1103  if I == 3, out_val = 3; U_B_cv = U_B_it; V_B_cv= V_B_it; S_B_cv = S_B_it; end
1104  % Output Iter. Ref. or Ref.
1105
1106  end
1107  %------------------------%
1108  % END: CONVERGENCE TESTS %
1109  %------------------------%
1110
1111  %--------------------------------------%
1112  % BEGIN: COMPUTE REFINED RITZ ITERATION %
1113  %--------------------------------------%
1114
1115  function [u,rho,v_min,rconv] = refined_ritz(D_ritz,R,alpha,B,maxitref,Bsz,k,method)
1116  % Computes the Iterative refined Ritz values and vectors.
1117  %
1118  % INPUT:
1119  %
```

```
1120 %   D_RITZ  - (K x 1) vector of approx. singular values.
1121 %        R  - real number - norm of residual vector F.
1122 %   ALPHA   - real number - last diagonal element of B -> B(Bsz,Bsz).
1123 %      B    - Bsz x Bsz bidiagonal matrix
1124 % MAXITREF  - Integer indicating the maximum number of iterations for the iterative
1125 %             Refined Ritz values.
1126 %             set to 1 and the refined Ritz are computed - used in  convergence testing.
1127 %     BSZ  -  Integer indicates the size of the  tridiagonal matrix.
1128 %     K    - number of Iterative refined Ritz values/vectros to compute
1129 %   METHOD -   NOR -  Equivalent eigenvalue system
1130 %                     AP_m = Q_mB_m  and A'Q_m = P_m B_m' + f_me_m'
1131 %                     Equivalent eigenvalue system
1132 %                     A'AP_m = P_m B_m'B_m + B_m(m,m)f_me_m'
1133 %                     Compute iterative refined Ritz on [B'*B; 0 B_m(m,m)*norm(f)]
1134 %              AUG -  Equivalent eigenvalue system
1135 %                     [0 A; A' 0][Q_m 0; 0 P_m] = [Q_m 0; 0 P_m][0 B; B' 0]+[ 0 0; f_me_m' 0]
1136 %                     Compute iterative refined Ritz on [0 B; B' 0; norm(f) 0]
1137 %
1138 % OUTPUT:
1139 %      U   - (BSZ x K) matrix of left singular vectors:
1140 %            METHOD
1141 %               NOR =>  U = B*V_min; U = U/norm(U);
1142 %               AUG =>  U = V_min(1:Bsz/2,Bsz); U = U/norm(U);
1143 %     RHO  - (K x 1) vector of iterative refined Ritz values.
1144 %   V_MIN  - (BSZ x K) matrix of right singular values
1145 %            METHOD
1146 %               AUG =>  V_MIN = V_MIN(Bsz/2+1:Bsz,Bsz); V_MIN = V_MIN/norm(V_MIN);
1147 %   RCONV  - Integer indicate if all K iterative refined Ritz converge.
1148 %          - 0 not all converged - do not use
1149 %          - K all converged
1150 %
1151 %
1152 %  DATE MODIFIED: 5/10/2021
1153 %  VER:  1.0
1154
1155 % Set up the augmented matrix [B'*B; 0 alpha*R]
1156 if strcmp(method,'NOR')
1157    B_aug = zeros(Bsz+1,Bsz); B_aug(1:Bsz,1:Bsz) = B'*B;
1158    B_aug(Bsz+1,Bsz) = alpha*R;
1159
```

```
1160     % Initialize rho value.
1161     rho = D_ritz.^2;
1162
1163 else
1164
1165     % Set up the augmented matrix [0 B; B' 0; R 0]
1166     B_aug = [zeros(Bsz) B; B' zeros(Bsz); zeros(1,2*Bsz)];
1167     B_aug(2*Bsz+1,Bsz) = R;
1168
1169     % Initialize rho value.
1170     rho = D_ritz;
1171
1172 end
1173
1174
1175 % Initialize values.
1176 s_min = zeros(k,1);           % Intialize min. singular values of B_aug.
1177 v_min = zeros(Bsz,k);         % Intialize right singular vectors of B_aug.
1178 u     = zeros(Bsz,k);         % Intialize left singular vectors of B_aug.
1179 rconv = zeros(k,1);           % Intialize convergence.
1180 rho_0 = rho;                  % Used to check for convergence during iterations.
1181 sqrteps = sqrt(eps);          % square root of machine precision - eps
1182 v_zero = zeros(Bsz,1);        % Set starting vector to zero.
1183
1184 % Change size for augmented system.
1185 if strcmp(method,'AUG'), Bsz = 2*Bsz; end
1186
1187 % Compute k number of iterative refined Ritz values/vectors.
1188 % Compute in reverse order. Less likely k, k-1, .. converge - avoids
1189 % uneeded iteration. Exit if any iterative refined Ritz fail to converge.
1190 for j = k:-1:1
1191
1192     %  Set difference in rho values to test for stagnation.
1193     diff_rho_0 = -1; v_min_0 = v_zero;
1194
1195     % Iteratation to compute the iterative refined Ritz values/vectors
1196     for i=1:maxitref
1197
1198       % Compute the SVD of  B_aug - rho* I
1199       [U,S,V] = svd((B_aug-rho(j)*eye(Bsz+1,Bsz)),0);
```

```
1200
1201        % Need the smallest singular triplet of B_aug - rho* I. Matlab
1202        % returns order of singular values largest to smallest.
1203        if strcmp(method,'NOR')
1204           s_min(j) = S(Bsz,Bsz); v_min(:,j) = V(:,Bsz);
1205
1206           % Compute the new rho = v_min'*B'*B*v_min = ||B*v_min||^2
1207           rho(j) = norm(B*v_min(:,j))^2; diff1 = 0;
1208
1209        else
1210
1211          s_min(j) = S(Bsz,Bsz);
1212          v_min(:,j) = V(Bsz/2+1:Bsz,Bsz); v_min(:,j) = v_min(:,j)/norm(v_min(:,j));
1213
1214          % Compute the new rho = V(:,Bsz)'*[0 B; B' 0]*V(:,Bsz)
1215          rho(j) = V(:,Bsz)'*B_aug(1:Bsz,:)*V(:,Bsz);
1216          % check norm singular vectors are close to 1/sqrt(2)
1217          diff1 = abs(norm(V(Bsz/2+1:Bsz,Bsz)) - 1/sqrt(2));
1218
1219        end
1220
1221        % Compute the difference of previous rho to check for convergence
1222        den = max([rho_0(j),rho(j), eps]);
1223        diff_rho = abs( (rho_0(j) - rho(j))/den );
1224
1225        % Check for convergence
1226        if ((((diff_rho < eps && abs(V(:,Bsz)'*U(1:Bsz,Bsz)) < sqrteps)...
1227           || s_min(j) < eps || norm(abs(v_min_0) - abs(v_min(:,j))) < eps) && diff1
1228           < sqrteps) || maxitref == 1)
1229              rconv(j) = 1;
1230              if strcmp(method,'NOR')
1231                 u(:,j) = B*v_min(:,j);
1232                 u(:,j) = u(:,j)/norm(u(:,j));
1233                 rho(j) = sqrt(rho(j));
1234
1235              else
1236
1237                 u(:,j) = V(1:Bsz/2,Bsz);
1238                 u(:,j) = u(:,j)/norm(u(:,j));
1239
```

```
1240              end
1241              break;
1242        end
1243
1244        % Check for stagnation to avoid too many unecessary iterations. Care
1245        % must be taken to avoid the situation where rho(k) is still changing
1246        % very slightly at first and then an increase in change later. An
1247        % early termination due to stagnation with no converegnce
1248        % may avoid increase in change later that converges.
1249        if abs(diff_rho_0 - diff_rho)/max(diff_rho,eps) < eps,  break; end  % stagnation
1250        if i>= 10 && mod(i,10)==0 % update every 10 iterations to avoid early termination.
1251           diff_rho_0 = diff_rho;
1252        end
1253
1254       % Update rho_0 and v_min_0 to test convergence.
1255       rho_0(j) = rho(j); v_min_0 = v_min(:,j);
1256      end
1257
1258      if rconv(j) == 0, rconv = 0; return; end
1259
1260  end
1261
1262  % After K iteration finish check to see if *all* K iterative refined Ritz values
1263  % have all converged. Reset rconv to return an integer value, rconv = k all
1264  % converged and rconv = 0, not all converged.
1265
1266  if all(rconv == 1), rconv = k; else rconv = 0; end
1267
1268  end
1269
1270  %-------------------------------------%
1271  % END: COMPUTE REFINED RITZ ITERATION %
1272  %-------------------------------------%
1273
```